O'REILLY®

TURING 图灵程序设计丛书



Python数据处理

Data Wrangling with Python

全面掌握用Python进行爬虫抓取以及数据清洗与分析的方法，轻松实现高效数据处理

[美] Jacqueline Kazil　Katharine Jarmul 著
张亮　吕家明 译

中国工信出版集团　人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

译者介绍

张亮（hysic）

毕业于北京大学物理学院，爱好机器学习和数据分析的核安全工程师。

吕家明

2016年毕业于哈尔滨工业大学，现就职于腾讯，从事搜索、Query分析等相关工作，熟悉大规模数据下的数据挖掘和机器学习实践。

延展阅读

Python基础教程（第2版•修订版）
978-7-115-35352-8

Python编程：从入门到实践
978-7-115-42802-8

Python网络数据采集
978-7-115-41629-2

流畅的Python
978-7-115-45415-7

干净的数据
978-7-115-42047-3

Python数据分析实战
978-7-115-43220-9

洞悉数据：用可视化方法发掘数据真义
978-7-115-41470-0

图灵程序设计丛书

Python数据处理

Data Wrangling with Python

[美] Jacqueline Kazil　Katharine Jarmul 著
张亮　吕家明 译

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北　　京

图书在版编目（CIP）数据

Python数据处理 / (美) 杰奎琳·凯泽尔
(Jacqueline Kazil) , (美) 凯瑟琳·贾缪尔
(Katharine Jarmul) 著 ; 张亮, 吕家明译. -- 北京 :
人民邮电出版社, 2017.7
（图灵程序设计丛书）
ISBN 978-7-115-45919-0

Ⅰ. ①P… Ⅱ. ①杰… ②凯… ③张… ④吕… Ⅲ. ①
软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2017)第126120号

内 容 提 要

本书采用基于项目的方法，介绍用 Python 完成数据获取、数据清洗、数据探索、数据呈现、数据规模化和自动化的过程。主要内容包括：Python 基础知识，如何从 CSV、Excel、XML、JSON 和 PDF 文件中提取数据，如何获取与存储数据，各种数据清洗与分析技术，数据可视化方法，如何从网站和 API 中提取数据。

本书适合数据处理工作相关人员。

◆ 著　　[美] Jacqueline Kazil　Katharine Jarmul
译　　张　亮　吕家明
责任编辑　岳新欣
执行编辑　魏书莉
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　印刷

◆ 开本：800×1000　1/16
印张：24.75
字数：594千字　　　　2017年 7 月第 1 版
印数：1－4 000册　　　2017年 7 月北京第 1 次印刷
著作权合同登记号　图字：01-2016-9369号

定价：99.00元
读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

本书赞誉

“所有新手数据科学家、数据工程师或其他技术方面的数据专家都应该读一读这本实践指南。数据处理领域正需要这样一本书，真希望我第一次开始用 Python 处理数据时就能有它指导。”

——Tyrone Grandison 博士，Proficiency Labs Intl. CEO

“数据处理不仅仅是编写代码，还包括更多内容，这本精心编写的书可以告诉你需要知道的一切内容。在新闻业需要更多数据专家的时代，这本书是循序渐进的宝贵资源。”

——Randy Picht，密苏里大学新闻学院 Donald W. Reynolds 新闻研究所执行理事

“很少有学习资源能够像这本书一样既全面又通俗易懂。它不仅介绍了你需要知道的内容，还阐释了其原因及学习方法。无论你是数据新闻业的新手，还是想要扩展自己的能力，Katharine 和 Jacqueline 的这本书都是必备的。”

——Joshua Hatch，《高等教育纪事报》与《慈善纪事报》数据与交互高级编辑

“这是一个很棒的概论课程，讲述了我们用数据讲故事时所做的一切，（真的是一切！）既包括了基础知识，也涵盖了最新技术。强烈推荐！”

——Brian Boyer，美国全国公共广播电台（NPR）可视化编辑

“这是一本实用的、通俗易懂的指南，你可以从中学习一些常见的不得不用代码完成的任务：查找、提取、整理和检查数据。”

——Chrys Wu，技术专家

“经常有记者问我：‘我很擅长使用电子表格，但下一步应该学些什么？’这本书给出了一

个很有价值的答案。虽然这本书不仅仅面向新闻业的读者，但它给出了一条清晰的路径，对于任何使用电子表格并且想知道如何提高技能的人来说，都可以沿着这条路径来学习获取、清洗和分析数据的方法。它涵盖了所有内容，从如何加载并检查文本文件到自动化屏幕抓取，再到执行数据分析与结果可视化的新的命令行工具。

“我曾经使用陈旧的方式来分析数据并寻找其中的意义：首先使用电子表格，然后转向关系型数据库和绘图程序。它们仍然是很有用的工具，但都没有充分利用自动化功能，让用户能够处理更多数据并复制其工作。它们也不能与互联网上的各种数据无缝连接。在这些工具旁边还需要添加上一种编程语言。虽然我现在已经使用 Python 和其他语言一段时间了，但这种使用漫无计划，并不系统。

“无论是数据处理还是工具的复杂性，在过去 20 年中都在不断发展，这使得寻找一套常用技术更为重要。不断增长的可用数据（结构化的和非结构化的）以及可以用于存储和分析的数据量，都改变了数据分析的可能性：许多困难的问题现在变得更容易回答了，之前看起来不可能的一些问题也已能力可及。我们需要一种‘胶水’，可以将数据生态系统的各个组成部分，从 JSON API 到数据过滤与清洗，再到创建图表来讲故事，全部连接在一起。

“在这本书中，这种‘胶水’就是 Python 及其用于处理数据的强大的工具和库。如果你一直感觉电子表格（甚至关系型数据库）无法回答你想要提出的问题，或者除这些工具之外你已经准备进一步学习，那么这本书非常适合你。我一直在等待这本书的出现。”

——Derek Willis，ProPublica 新闻应用开发者，OpenElections 联合创始人

目录

前言

欢迎打开这本书。在本书中，我们将会让你的数据处理技术更上一层楼，不再只是使用电子表格，而是可以利用 Python 编程语言，将噪声数据轻松快速地转换成可用的报告。Python 语法简单，上手很快，人人都可以用 Python 编程。

想象一下，你每周都要手动重复同一过程，比如从多个来源复制数据并粘贴到一个电子表格中，用于后续处理。这项任务可能每周都需要花费一两个小时。但当你用脚本把这项任务自动化之后，它可能只需要 30 秒就可以完成！这会节省你的时间，让你做点其他事情，或者把更多的任务自动化。再想象一下，之前你无法处理某种格式的数据，但你现在能对数据进行格式转换，完成之前无法完成的任务。但在完成本书的 Python 练习后，你应该可以更有效地从之前认为不可用的数据（过于混乱，或者数据量过大）中采集信息。

我们将带领你完成数据获取、数据清洗、数据呈现、数据规模化和自动化的过程。我们的目标是教你学会轻松处理数据的方法，这样你就可以花更多的时间专注于内容和分析。我们将克服现有工具的局限，将手动处理过程替换为简洁、易读的 Python 代码。读完这本书后，你能够将数据处理过程自动化，定期执行文件编辑和清洗任务，获取并解析你之前无法获取的数据，还能处理数据量更大的数据集。

采用基于项目的方法，每一章的复杂度会逐渐增加。我们建议你跟随本书的节奏，将书中的方法应用到自己的数据集上。如果你没有一个特定的项目或研究，也可以使用本书线上的样本数据集。

目标读者

本书针对的是那些不想用桌面工具来探索数据处理的人。如果你精于 Excel，想进一步提升数据分析水平，本书将助你一臂之力！如果你之前学过其他语言，想用 Python 学习数据处理，也会发现本书非常有用。

如果你遇到不懂的问题，建议你联系我们，这样我们可以改进书的内容。你也应该使用互联网搜索或在线提问（在线提问有一些方法和技巧，请参考 https://www.propublica.org/nerds/item/how-to-ask-programming-questions）来补充学习。我们在附录 E 中介绍了一些调

试的技巧，你可以翻到那里看一下。

不适合阅读本书的读者

本书肯定不适合经验丰富的 Python 程序员，他们已经知道数据处理任务需要用到哪些库和技术。（对于这些人，我们推荐 Wes McKinney 写的《利用 Python 进行数据分析》。）如果你是经验丰富的 Python 开发者，或使用过 Scala、R 等其他具有数据分析能力的语言，本书可能也不适合你。但如果你是经验丰富的 Web 语言开发者，使用的 PHP、JavaScript 等语言本身缺乏数据分析能力，那么本书可以通过数据处理来教你 Python 的知识。

本书结构

本书的结构沿循一般数据分析项目或故事的整个生命周期。首先提出一个问题，然后获取数据、清洗数据、探索数据、传达数据中的发现、扩展到更大的数据集，最后将整个过程自动化。这种方法可以让你从简单的问题逐步过渡到更复杂的问题和研究。我们会先讲传达数据中发现的基本方法，然后再讲数据采集的高级技巧。

如果对某些章节的内容比较熟悉，你也可以将本书当作参考，或者跳过那些章节。但我们建议你大致浏览一下每一章节的内容，确保没有错过新的资源与技术。

什么是数据处理

数据处理是指将杂乱的或未加工的数据源转换成有用的信息。先寻找原始数据源，并判断其价值：这些数据集的数据质量有多好？它们与你的目标是否相关？能否找到更好的数据源？在对数据进行解析与清洗后，数据集变得可用，这时你可以利用工具和方法（如 Python 脚本）来帮你分析数据，并以报告的形式展示结果。这样你可以将无人问津的数据变得清晰可用。

遇到困难怎么办

不必担心——每个人都会遇到困难！把编程过程看作困难重重的一连串事件。作为一名开发者和数据分析人员，遇到困难，然后解决问题，你会学到知识，并得到成长。大多数人并非掌握了编程，而是掌握了解决困难的方法。

解决困难都有哪些技巧呢？首先，你可以利用搜索引擎尝试寻找答案。通常情况下，你会发现许多人已经遇到过相同的问题。如果找不到有用的答案，你可以在网上提问。我们在附录 B 中给出了一些优质的在线资源和线下资源。

提出问题是很难的。但无论处于学习的哪个阶段，都不要害怕向大型编程社区求助。本书作者 Jackie 早期在公共论坛上问了一个关于编程的问题（http://stackoverflow.com/questions/3329943/git-branch-fork-fetch-merge-rebase-and-clone-what-are-the-differences），之后被许多人引用过。像你一样的新手程序员硬着头皮问了一个可能很傻的问题，但之后却帮助了许多人，这种感觉是很爽的。

在网上发布问题之前，我们还推荐你阅读“如何提问”（https://www.propublica.org/nerds/item/how-to-ask-programming-questions）。这篇文章讲了许多方法，可以帮你正确地描述问题，这样其他人才能最大限度地帮助你。

最后，有时你还需要现实生活中的额外帮助。可能是你的问题涉及面太广，在网站或邮件列表里不方便问答。也可能是你的问题有些偏重哲学，或者需要不同方法之间的对比或修改。无论是哪种情况，你都可以在当地 Python 小组中找到能回答你问题的人。想要找到当地的线下聚会，你可以试试 Meetup 网站（http://www.meetup.com/）。关于如何找到有帮助且乐于助人的社区，你会在第 1 章中读到更多详细信息。

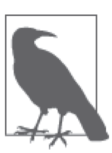
排版约定

本书使用了下列排版约定。

- 楷体
  表示新术语和重点强调的内容。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键词等。
- 斜体等宽字体（*`constant width italic`*）
  表示应该替换成用户输入的值，或根据上下文替换的值。



该图标表示提示或建议。



该图标表示一般性说明。



该图标表示警告或警示。

使用代码示例

我们在 GitHub 上建了一个数据仓库（https://github.com/jackiekazil/data-wrangling）。在这个仓库中，你会找到我们在本书中使用的数据，以及一些代码示例，让你可以跟上本书的节奏。如果你在仓库中发现任何问题或者有任何疑问，请提交一个 issue（issue 提交地址：https://github.com/jackiekazil/data-wrangling/issues）。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序和文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可。销售或分发 O’Reilly 图书的示例光盘则需要获得许可。引用本书中的示例代码来回答问题无需获得许可。将书中大量示例代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如：“*Data Wrangling with Python* by Jacqueline Kazil and Katharine Jarmul (O’Reilly). Copyright 2016 Jacqueline Kazil and Kjamistan, Inc., 978-1-4919-4881-1.”

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online

Safari Books Online（http://safaribooksonline.com/）是应运而生的数字图书馆，它同时以图书和视频的形式出版世界顶级技术和商业作家的专业作品（https://www.safaribooksonline.com/explore/）。

技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于企业（https://www.safaribooksonline.com/enterprise/）、政府（https://www.safaribooksonline.com/government/）、教育机构（https://www.safaribooksonline.com/academic-public-library/）和个人，Safari Books Online 提供各种产品组合和灵活的定价策略（https://www.safaribooksonline.com/pricing/）。

用户可通过一个功能完备的数据库检索系统访问 O’Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他数百家出版社（https://www.safaribooksonline.com/our-library/）的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，请访问我们的网站（https://www.safaribooksonline.com/）。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那里找到本书的相关信息，包括勘误表、示例以及其他信息。本书的网站地址是：http://bit.ly/data_wrangling_w_python。[1]

对于本书的评论和技术性问题，请发送电子邮件到：bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia

致谢

我们想要感谢本书的编辑 Dawn Schanafelt 和 Meghan Blanchette，感谢他们给予极大的帮助和付出以及了不起的工作——没有你们就不可能有这本书的问世。我们还想感谢本书的技术编辑 Ryan Balfanz、Sarah Boslaugh、Kat Calvin 和 Ruchi Parekh，感谢他们帮助完成代码示例以及思考本书受众。

Jacqueline Kazil（Jackie）想要感谢她的丈夫 Josh 支持她写作本书——无论是鼓励的话语还是纸杯蛋糕。如果没有他的全力支持，房子可能就散架了。她还想要感谢 Katharine（Kjam）与她合作。没有 Kjam 就没有这本书。Jackie 很高兴在分别多年之后再次有机会与 Kjam 合作。最后，Jackie 想感谢自己的母亲 Lydie，教给她写作本书所必须的许多技能（除了英语）。

Katharine Jarmul 想要拥抱并特别感谢她的伴侣 Aaron Glenn，感谢他无数个小时的自言自语、反复阅读、争论 Unix 首字母是否应该大写，以及在她写作时制作美味的意大利面。她还要感谢她的父母和公婆，感谢他们对她无数次图书校正和响铃的忍耐。她还要感谢德国的 Hoffmann 夫人，感谢她在讨论本书的过程中所付出的极大耐心。

电子书

扫描如下二维码，可购买本书电子版。

注 1：本书中文版勘误可到 ituring.cn/book/1819 查看和提交。——编者注

第 1 章

Python简介

无论你是一名记者、分析师，还是初出茅庐的数据科学家，选择这本书可能是因为你想学习如何用编程来分析数据，得出结论，并将结论清楚地传达给别人。你可能会用报告、图表或归纳统计的方式来展示你的结论。重要的是，你想讲述一个故事。

传统的故事讲述或新闻报道往往使用单一的故事来描述总体结论或趋势。在这种故事中，数据成为了相对次要的部分。然而，其他讲故事的人，比如 Christian Rudde［*Datacylsm*（http://dataclysm.org/）的作者，OkCupid 的创始人之一］认为数据本身应该是故事的重点。

首先，你需要确定想要研究的主题。你可能对研究不同人或群体的沟通习惯感兴趣，这时你可以从一个具体的问题入手，例如在网络上被人们广为分享的信息都有哪些特点。又或许你可能对棒球的历史统计数据感兴趣，并想弄清楚一个问题：这些数据能否表明棒球运动随时间发生了变化。

确定了感兴趣的领域之后，你需要寻找数据，以进一步探索这一主题。想研究人类行为，你可以从 Twitter API（https://dev.twitter.com/overview/api）中获取数据，研究人们在 Twitter 上分享的内容。如果想深入研究棒球历史，你可以使用 Sean Lahman 的棒球数据库（http://www.seanlahman.com/baseball-archive/statistics/）。

Twitter 和棒球数据集都属于综合的大型数据集。为了回答你的具体问题，应把这些数据集分成便于管理的小块，然后进行筛选和分析。有时，较小的数据集也同样有趣有料，尤其当你的主题是关于本地或区域问题时。让我们来看一个例子。

在写这本书的时候，本书的一位作者读到一篇关于她的公立高中母校[1]的文章（http://www.foxnews.com/on-air/fox-and-friends/blog/2014/05/20/manatee-high-school-charges-200-row-

注 1：美国公立高中是政府办的学校，财政主要由当地社区的税收资助。孩子们在这里上学受教育是免费的，或只需花父母很少的钱。

prime-seating-graduation）。文中写道，这所高中开始向每名即将毕业的学生收取 20 美元，并且对于毕业典礼上最好的几排座位，每排收取 200 美元。

据当地新闻报道："学区今年财政困难，撤出了 3400 美元的资助。新的收费是为了支付海牛高中约 12 000 美元的毕业典礼费用而采取的措施之一。"

这篇文章解释了毕业典礼花费远高于学区预算的原因。然而文中并没有解释学区今年为何无法像往年一样进行资助。所以我们仍有疑问：海牛郡学区的财政为何如此困难，以至于无法像往年一样资助毕业班级？

调查之初所提出的疑问，往往会引出更深入的疑问，直指问题的本质。例如，学区把钱都花到了什么地方？学区的开支模式近些年发生了哪些变化？

确定了具体的主题和要回答的问题，可以让我们明确需要寻找哪些数据。在提出上述问题后，我们要寻找的第一个数据集就是海牛郡学区的开支和预算数据。

在继续讨论之前，我们先简单回顾一下整个过程，从开始确定问题一直到最终的故事（见图 1-1）。

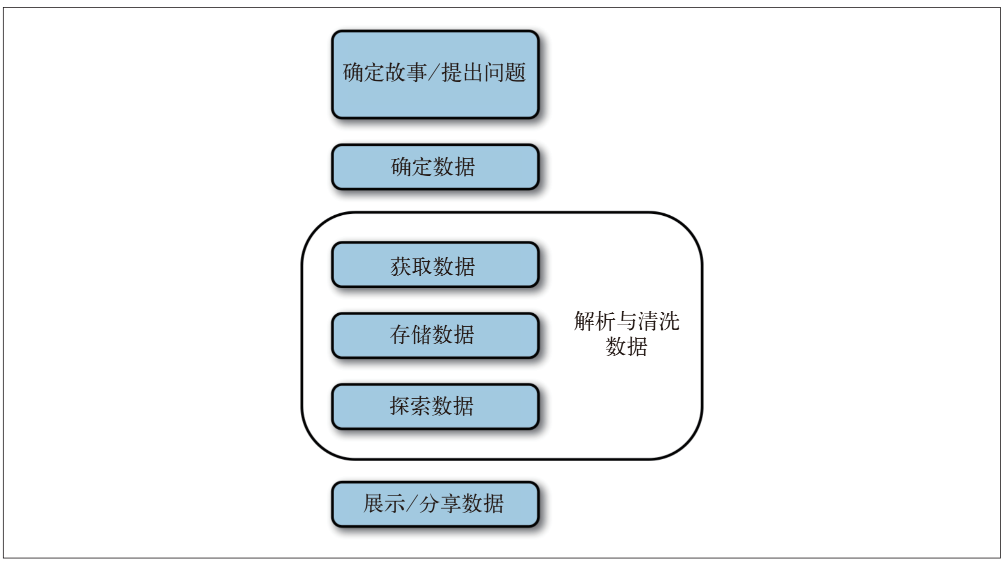
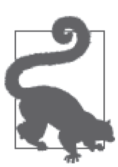


图 1-1：数据处理过程

一旦确定了问题，你就可以开始提出有关数据的问题，例如：哪些数据集最能帮助我讲好故事？哪些数据集能帮助我深入探索这一主题？总的主题是什么？哪些数据集与这些主题相关？谁会记录并保存这些数据？可以公开获取这些数据集吗？



在开始讲故事时，你应该专心研究想要回答的问题。然后你可以找出哪些数据集对你是最有价值的。在这个初始阶段，不要过分纠结于分析数据的工具或数据处理过程。

寻找数据集

如果使用搜索引擎来寻找数据集，你不一定总能找到最合适的。有时你需要在一个网站中仔细寻找数据。即使数据难以找到或难以获取，也一定不要放弃！

如果你的主题出现在一份调查或报告中，或者某个特定的机构或组织可能会收集关于这一主题的数据，你应该去查找联系人号码，并联系研究人员或组织。礼貌而直接地询问如何能够访问他们的数据。如果数据集属于政府部门（联邦、州或地方），那么根据（美国）信息自由法案（https://en.wikipedia.org/wiki/Freedom_of_Information_Act_(United_States)），你也许可以合法地直接获取这些数据。我们将在第 6 章中更全面地讨论数据获取。

一旦确定了需要的数据集并获取了这些数据，你需要将其转换成可用的格式。在第 3、4、5 章中，你将会学习各种方法，用于通过编程的方式获取数据并对其进行格式转换。第 6 章讲的是在获取数据时如何与不同的人打交道，并稍稍提到合法性的问题。在第 3 章至第 5 章中，我们还将介绍如何从 CSV、Excel、XML、JSON 和 PDF 文件中提取数据。在第 11、12、13 章中，你将会学习如何从网站和 API 中提取数据。

> 如果你不认识某些缩写词，别担心！在遇到这些缩写词时我们会详细解释，也会一并解释其他你可能不熟悉的技术术语。

在取得数据并转换格式后，你将开始初步的数据探索。你将会探寻数据中可能隐藏的所有故事，同时判断哪些故事有用，哪些故事可以舍弃。你还可以把数据分组，观察各组之间的变化趋势。然后还可以将数据集合并，寻找其中的关联，发现更大的趋势，并发现潜在的矛盾之处。在这一过程中，你将学习如何清洗数据，发现并解决隐藏在数据集中的问题。

在第 7 章和第 8 章中你将学习如何解析与清洗数据。你既可以使用 Python，也可以探索其他开源工具。在讨论可能遇到的数据问题时，你将学会应该选择哪种方法清洗数据：是写一个清洗脚本，还是使用现成的方法。在第 7 章中，我们将讲到如何处理常见的错误，如重复记录、离群值与格式化问题。

在确定你要讲述的故事，清洗并处理数据之后，我们将探讨如何利用 Python 来展示数据。你将学习用各种形式来讲故事，并比较不同的发布方式。在第 10 章中，你将学习在网站上展示与组织数据的基本方法。

第 14 章将使数据分析过程规模化，让你可以在更短的时间内处理更多的数据。我们将对存储与获取数据的方法进行分析，并研究在云中使数据规模化的方法。

第 14 章还将讨论如何完成一次性的项目，并使其能够自动完成。通过自动化过程，你可以将一次性的特殊报告变成年度报告。这种自动化可以让你专注于改善讲故事的过程，继续讲下一个故事，或者至少续杯咖啡。本书主要使用的工具是 Python 编程语言。在我们讲故事过程的每一步，从最初的探索直到标准化与自动化，都会用到 Python。

1.1　为什么选择Python

有那么多种编程语言，本书为什么选择使用 Python？你可能听说过以下这些语言中的一种或多种：R、MATLAB、Java、C/C++、HTML、JavaScript 和 Ruby，这取决于你的专业背景。这些语言都有一种或多种主要用途，有些还可以用于数据处理。你也可以用 Excel 等程序进行数据处理。用 Excel 和 Python 编程，通常会得到相同的结果，但其中一种语言的效率更高。然而有时候，像 Excel 这样的程序无法完成任务。我们选择 Python 而不是其他语言，是因为 Python 很容易上手，处理数据的过程也简单明了。

如果你想了解 Python 和其他语言的技术性区别，可查阅附录 A。看完里面的解释，你就可以告诉其他分析师或开发人员你使用 Python 的原因。我们相信作为开发新人，你会从 Python 的易用性中受益，同时我们希望本书能成为你的数据处理工具箱中有参考价值的一本书。

Python 除了上述作为语言的优点，Python 社区还是最开放和最乐于助人的社区之一。世上没有完美的社区，但 Python 社区为新人创造了一个良好的环境：有时会有本地的教程、免费课堂、线下聚会等，有时还会举办大型会议，人们聚在一起解决问题并分享知识。

拥有大型社区的好处显而易见——有人能回答你的问题，有人能为你的代码或模块的结构出谋划策，有人能让你学习借鉴，还有共享代码供你在其之上构建自己的代码。想了解更多的内容，可查阅附录 B。

社区因成员的支持而存在。刚开始使用 Python 时，你从社区中得到的帮助会多于你的付出。但即使你不是专家，也可以为社区做很多贡献。我们鼓励你分享你的问题和解决方法。这会帮到下一个遇到相同问题的人，你也可能会在开源工具中发现需要处理的 bug。

> 在现阶段，你是从新鲜视角看待问题，而 Python 社区的很多成员已经不再这样看待问题了。当开始敲 Python 代码时，你应该把自己当作编程社区的一员。你的贡献与那些有 20 年编程经验的人一样重要。

言归正传，我们开始讲 Python。

1.2　开始使用Python

编程的第一步是最难的。（其难度和你学习走路时迈的第一步没有什么不同！）回想一下你培养新爱好或学习新运动的时候。在入门 Python（或任何其他编程语言）时，你将会遇到类似的焦虑和障碍。或许你很幸运，有一个优秀的导师帮你迈出第一步。如果没有的话，你可能已经历过类似的挑战。无论你如何迈出第一步，一定要记住，这个阶段往往是最艰难的。

> 我们希望本书可以成为你的入门指南，但它无法替代优秀的导师或使用 Python 的丰富经验。在本书中，我们提供了一些资源和网站信息，以便你在遇到书中没有讲到的问题时有所参考。

为了避免浪费太多时间进行大量配置和高级安装，我们的 Python 环境将采用最简单的初始安装。在接下来的几节中，我们将选定 Python 版本，安装 Python 以及一个可以帮助我们使用外部代码和库的工具（pip），并安装代码编辑器，以便编写并运行代码。

1.2.1 Python版本选择

你需要选择使用哪个版本的 Python。Python 版本实际上指的是 Python 解释器的版本。解释器让你可以在计算机上读写并运行 Python。维基百科（https://en.wikipedia.org/wiki/Interpreter_(computing)）是这样描述解释器的：

> 在计算机科学中，解释器是一种计算机程序，可以直接运行用编程语言或脚本语言编写的指令，而无需事先将其编译成机器语言程序。

没有人会让你背诵这个定义，所以即使你没有完全理解也不必担心。本书的作者之一 Jackie 刚开始编程的时候，无法理解入门书中“批处理编译”（batch compiling）这一概念，所以她就卡在这一部分无法继续读下去。如果她连这都不懂，怎么能编程呢？我们在后面会讨论编译的话题，但是现在将上述定义总结如下：

> 解释器是读取并运行 Python 代码的计算机程序。

Python（或 Python 解释器）主要有两个版本：Python 2.X 和 Python 3.X。Python 2.X 的最新版本是 Python 2.7，这也是本书使用的 Python 版本。Python 3.X 的最新版本是 Python 3.5，这也是最新可用的 Python 版本。目前你可以认为 Python 2.7 的代码无法在 Python 3.4 中运行。专业的说法是，Python 3.4 破坏了*向后兼容*。

你可以写出同时兼容 Python 2.7 和 Python 3.4 的代码，但这既非本书的要求，也不是本书的重点。一开始就在这件事情上花费大量精力，就好比住在佛罗里达州的人担心如何在雪中开车一样[2]。有一天你可能会需要这项技能，但此时此刻这并不是你需要关注的重点。

有些读者可能会好奇我们为何决定使用 Python 2.7 而不是 Python 3.4。这个话题在 Python 社区中是很有争议的。Python 2.7 是被广泛使用的版本，而 Python 3.X 正在被逐步接受。我们之所以选择 Python 2.7，是想保证你能够找到容易阅读且容易获取的资源，并确保你的操作系统及服务支持你使用的 Python 版本。

> 本书中的很多代码在 Python 3 中也可以运行。如果你想在 Python 3 中运行其中一些示例，当然可以；但我们更希望你专注于学习 Python 2.7，在学完本书后再继续学习 Python 3。为了兼容 Python 3 需要修改代码，想了解这方面的更多内容，可以查看官方更改文档：https://docs.python.org/3.0/whatsnew/3.0.html。

在阅读本书的过程中，你会用到自己编写的代码，还会用到其他（牛）人编写的代码。这些外部代码大部分都可以在 Python 2.7 下运行，但可能无法兼容 Python 3.4。如果你正在使用 Python 3 的话，需要重写代码（如果对遇到的每一段代码都要进行重写和编辑，这会浪费大量的时间，你可能很难完成第一个项目）。

注 2：美国佛罗里达州又名“阳光之州”，一年四季很少下雪。——译者注

把你的第一段代码当作草稿，以后你可以回来进一步修订完善。现在我们开始来安装 Python。

1.2.2 安装Python

好消息是，Python 可以在所有操作系统上运行。坏消息是，不同操作系统的安装方法有所不同。按照 Python 编程的流行程度排序，我们将讨论两个主要的操作系统：Mac OS X 和 Windows。如果你用的是 Mac OS X 或 Linux，可能已经安装了 Python。想了解更详细的安装过程，我们推荐以你的 Linux 发行版加"Python 高级安装"（advanced Python setup）为关键词在 Web 上搜索，以寻求更多建议。



与 Windows 系统相比，在 OS X 系统和 Linux 系统上安装和运行 Python 要容易一些。想深入了解这种差异存在的原因，我们推荐阅读 Windows 和基于 Unix 的操作系统的历史。可以对照阅读这两篇文章：一篇是 Hadeel Tariq Al-Rayes 写的"Studying Main Differences Between Linux & Windows Operating Systems"（http://www.ijens.org/vol_12_i_04/126704-8181-ijecs-ijens.pdf），文中的观点支持 Unix；另一篇是微软的"Functional Comparison of UNIX and Windows"（https://technet.microsoft.com/en-us/library/bb496993.aspx）。

如果你用的是 Windows，应该也可以运行所有的代码；但是在 Windows 下可能还需要安装代码编译器和额外的系统库，以及设置环境变量。

为了安装 Python 并能正常使用，请根据你的操作系统安装提示，按照相应的步骤来操作。我们还将进行一系列的测试，确保一切正常运行，然后才能进入下一章。

1. Mac OS X系统

首先打开终端（http://en.wikipedia.org/wiki/Terminal_(OS_X)），一个让你可以和计算机进行交互的命令行界面。个人计算机（PC）刚刚出现时，你只能通过命令行界面与计算机交互。现在大多数人用的都是图形界面操作系统，因为这种操作系统访问更方便，用户也更加广泛。

有两种方法可以在计算机上找到终端。第一种方法是通过 OS X 系统的 Spotlight。点击 Spotlight 图标——屏幕右上角的放大镜——输入"Terminal"。然后选择出现在"应用程序"（Applications）类别旁边的图标。

选定之后，会弹出一个类似图 1-2 中的小窗口（请注意，你的 Mac OS X 版本不同，界面可能也会有所不同）。



图 1-2：利用 Spotlight 搜索终端

你还可以通过 Finder 启动终端。终端位于“实用工具”（Utilities）文件夹：应用程序（Applications）→实用工具（Utilities）→终端（Terminal）。

选择并启动终端后，你应该会看到类似图 1-3 的界面。

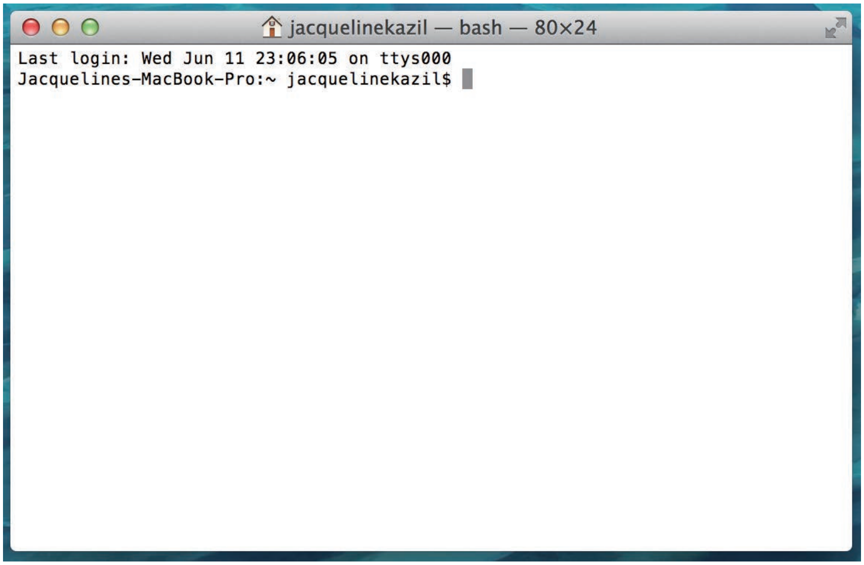


图 1-3：新打开的终端窗口

现在最好在适当的位置（比如在 Dock 中）创建终端的快捷方式，这样以后启动终端会比较方便。你只需在 Dock 中的终端图标上点击右键，依次选择“选项”（Options）和“在 Dock 中保留”（Keep in Dock），即可创建终端的快捷方式。运行本书中的每一个练习都需要启动终端。

大功告成。Mac 电脑预装了 Python，你无需再做其他事情。如果你想设置计算机使其能够使用后面的高级库，请查看附录 D。

2. Windows 8和Windows 10

Windows 并没有预装 Python，但有专门的 Python 安装程序（https://www.python.org/downloads/windows/）。你需要弄清楚你的 Windows 是 32 位还是 64 位（http://windows.microsoft.com/en-us/windows7/find-out-32-or-64-bit）。如果你用的是 64 位 Windows，需要从下载页面下载 x86-64 MSI 安装程序；如果不是 64 位的话，你可以下载 x86 MSI 安装程序。

下载好安装程序之后，只需双击它，按照提示一步步安装即可。我们建议选择为所有用户安装。点击靠近选项的方框选中全部，同时选择在硬盘上安装 Python 特性（见图 1-4）。

成功安装 Python 后，你需要将 Python 加入到环境设置中。这样你就可以在 cmd 工具（Windows 下的命令行界面）中与 Python 交互。要做到这一点，只需在计算机中搜索“环境变量”（environment variable）。选择“编辑系统环境变量”（Edit the system environment variables）选项，然后点击“环境变量…”（Environment Variables...）按钮（见图 1-5）。

将“系统变量”（System variables）列表下拉，选择 Path 变量，然后点击“编辑”（Edit）。如果列表中找不到 Path 变量，点击“新建”（New）创建一个新的 Path 变量。

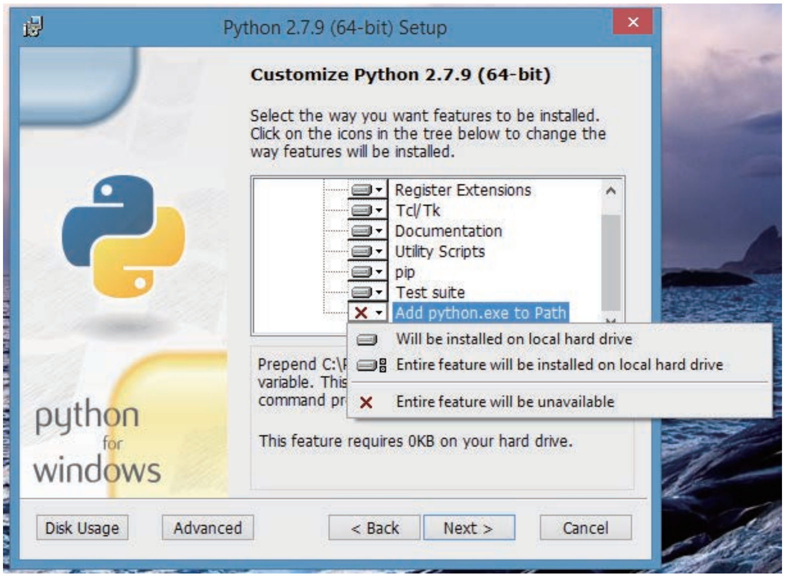


图 1-4：利用安装程序添加 Python 特性

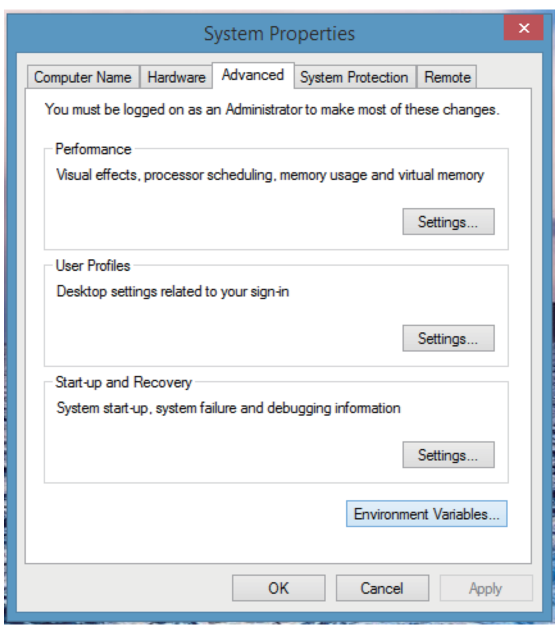


图 1-5：编辑环境变量

将“系统变量”（System variables）列表下拉，选择 Path 变量，然后点击“编辑”（Edit）。如果列表中找不到 Path 变量，点击“新建”（New）创建一个新的 Path 变量。

将下列内容添加到 Path 变量的末尾，注意每一个路径之间要用分号隔开（包括已有内容末尾的分号）：

```
C:\Python27;C:\Python27\Lib\site-packages\;C:\Python27\Scripts\;
```

Path 变量的末尾应该类似图 1-6 所示。编辑好环境变量之后，点击“确定”（OK）来保存设置。

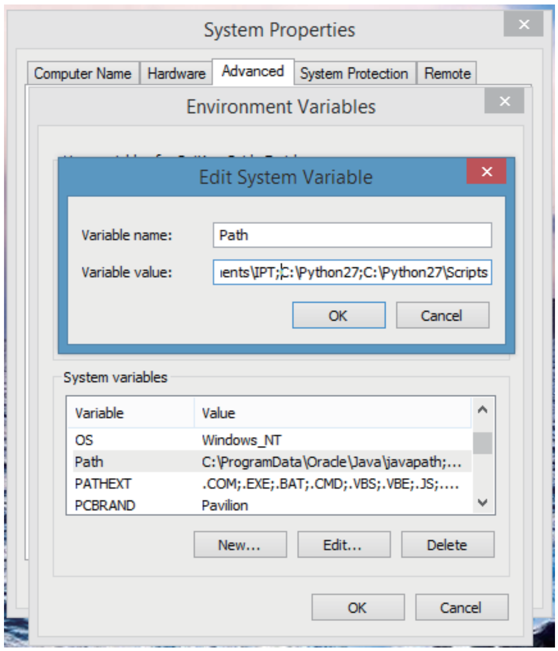


图 1-6：将 Python 加入 Path 系统变量

1.2.3 测试Python

现在你应该已经打开命令行（终端或 cmd[3]），准备启动 Python。在 Mac 上命令行的结尾应该是 $ 符号，在 Windows 上应该是 > 符号。在提示符后输入 python，然后按 Return（或 Enter）键：

```
$ python
```

一切正常的话，你应该会看到 Python 提示符（>>>），如图 1-7 所示。

注 3：想在 Windows 上打开 cmd 工具，只需搜索“命令提示符”（Command Prompt），或者打开“所有程序”（All Programs），依次选择“附件”（Accessories）和“命令提示符”（Command Prompt）。

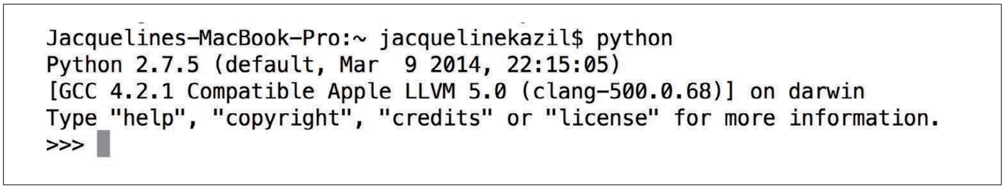


图 1-7：Python 提示符

对于 Windows 用户，如果没有出现 Python 提示符，请检查 `Path` 变量的设置是否正确（如前文所述），软件是否均已正确安装。如果你用的是 64 位版本，可能需要卸载 Python（可以用下载的 MSI 文件对 Python 安装进行修改、卸载和修复），并尝试安装 32 位版本。这样还不行的话，我们建议你将安装过程中遇到的具体错误信息放到网上去搜索。

`>>>` 与 `$` 或 `>` 的对比

Python 提示符与系统提示符（在 Mac/Linux 上是 `$`，在 Windows 上是 `>`）有所不同。初学者往往犯这样的错误：在默认的终端提示符后面输入 Python 命令，而在 Python 解释器中输入终端命令。两种情况都会报错。遇到报错时，记住上面这两种错误，检查确认你只在 Python 解释器中输入 Python 命令。

如果把本应输入到系统终端的命令输入到 Python 解释器中，可能会出现 `NameError`（命名错误）或 `SyntaxError`（语法错误）。而如果在系统终端中输入 Python 命令，可能会出现 bash 错误 `Command not found`。

Python 解释器启动时给出了几行有用的信息。其中一行显示的是我们正在使用的 Python 版本（图 1-7 中显示的是 Python 2.7.5）。这一信息在错误排查过程中非常重要，因为有些时候，某些命令或工具可以在这一版本的 Python 中正常运行，在另一版本中却无法正常运行。

下面我们用 `import` 语句来快速测试一下 Python 的安装是否成功。在 Python 解释器中输入下列代码：

```
import sys
import pprint
pprint.pprint(sys.path)
```

代码的输出应该是一个列表，列表里面是你计算机中的许多目录或位置。这个列表给出的是 Python 程序寻找 Python 文件的具体位置。当你想要排查 Python 的模块导入错误时，这一组命令十分有用。

下面是一个输出实例（你的计算机输出的列表可能会有所不同；还需要注意的是，为了适应本书的页面宽度，对某些内容进行了折行）：

```
['',
 '/usr/local/lib/python2.7/site-packages/setuptools-4.0.1-py2.7.egg',
 '/usr/local/lib/python2.7/site-packages/pip-1.5.6-py2.7.egg',
 '/usr/local/Cellar/python/2.7.7_1/Frameworks/Python.framework/Versions/2.7/
   lib/python27.zip',
 '/usr/local/Cellar/python/2.7.7_1/Frameworks/Python.framework/Versions/2.7/
```

```
  lib/python2.7',
 '/usr/local/Cellar/python/2.7.7_1/Frameworks/Python.framework/Versions/2.7/
  lib/python2.7/lib-tk',
 '/Library/Python/2.7/site-packages',
 '/usr/local/lib/python2.7/site-packages']
```

如果代码运行失败，你会得到错误信息。调试 Python 错误最简单的方法就是阅读错误信息。例如，如果你输入的是 `import sus` 而不是 `import sys`，你会看到以下输出：

```
>>> import sus
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named sus
```

阅读最后一行：`ImportError: No module named sus`。这句话的意思是出现了模块导入错误，因为 Python 中没有 `sus` 模块。Python 搜遍计算机中的文件，没有找到名为 `sus` 且可导入的 Python 文件或文件夹。

如果在敲本书代码时犯了拼写错误，可能会引发语法错误。在接下来的例子中，我们故意把 `pprint.pprint` 拼错，输入的是 `pprint.print(sys.path())`：

```
>>> pprint.print(sys.path())
  File "<stdin>", line 1
    pprint.print(sys.path())
               ^
SyntaxError: invalid syntax
```

上面我们是故意输错的，但在本书的写作过程中，本书的一位作者真的输错了这行代码。你要习惯于在出现错误时排查并解决错误。你也应该承认，作为开发人员，错误是学习过程的一部分。我们希望你能对错误习以为常。你应该把错误看作一次学习的机会，可以让你学习 Python 和编程的新知识。

模块导入错误和语法错误都是代码开发过程中最常见的错误，并且也是最容易解决的。在遇到错误时，网络搜索引擎是帮你处理错误的得力助手。

在进入下一节之前，一定要退出 Python 编辑器。这样你就回到了终端或 cmd 的提示符界面。输入以下代码来退出 Python：

```
exit()
```

现在你的提示符应该变回了 `$`（Mac/Linux）或 `>`（Windows）。在下一章中我们将进一步研究 Python 解释器。接下来我们来安装一个叫作 pip 的工具。

1.2.4　安装pip

pip（http://pip.readthedocs.org/en/latest/）是用于管理 Python 共享代码和库的命令行工具。程序员经常会解决相同的问题，所以会分享他们的代码来帮助他人。这也是开源软件文化的一个重要组成部分。

Mac 用户可以通过在终端中运行下载的简单 Python 脚本来安装 pip（https://pip.pypa.io/en/latest/installing/#install-pip）。你需要与下载的脚本处于同一文件夹。例如，如果你将脚本

下载到 Downloads 文件夹中，你需要在终端中进入这个文件夹。在 Mac 上有一种简单的方法，按住 Command 键（Cmd），然后把 Downloads 文件夹拖入终端中。另一种方法是输入一些简单的 bash 命令（附录 C 中有对 bash 更全面的介绍）。首先在终端中输入以下命令：

```
cd ~/Downloads
```

这条命令将计算机的目录切换到主文件夹下的 Downloads 子文件夹。为了检查你是否位于 Downloads 文件夹下，在终端中输入以下命令：

```
pwd
```

这条命令可以在终端中显示你的当前工作目录，即你现在位于的文件夹。它的输出应该是类似下面这样的内容：

```
/Users/your_name/Downloads
```

如果输出没问题，你只需用这个命令就可以运行该文件：

```
sudo python get-pip.py
```

由于你运行的是 `sudo` 命令（你正在用特殊权限运行该命令，这样才可以在受限区域安装软件包），系统会提示你输入密码。你应该可以看到安装软件包的一系列提示信息。

> Windows 上可能已经安装了 pip（Windows 的 Python 安装包自带 pip）。要检查是否安装了 pip，你可以在 cmd 工具中输入 `pip install ipython`。如果系统报错的话，去下载 pip 安装脚本，用 `chdir C:\Users\`*YOUR_NAME*`\Downloads` 命令将目录切换到 Downloads 文件夹（把 *YOUR_NAME* 换成你计算机主目录的名字）。然后，你应该可以输入 `python get-pip.py` 来运行下载的文件。你需要管理员权限来确保正确安装。

在使用 pip 时，你的计算机在 PyPI（https://pypi.python.org/pypi）上搜索指定的代码包或代码库，将其下载到计算机中并安装。你无需使用浏览器来下载代码库，省去了许多麻烦。

安装工作已经基本完成。最后一步是安装代码编辑器。

1.2.5 安装代码编辑器

在写 Python 代码时需要一个代码编辑器，因为 Python 需要特殊的间距、缩进和字符编码才能正常运行。有多种代码编辑器可供选择。本书的一位作者使用 Sublime。它是免费的，但在你使用一段时间后会建议你象征性地支付一点费用，以支持当下和未来的开发。你可以在 http://www.sublimetext.com 下载 Sublime。另一款完全免费且跨平台的文本编辑器是 Atom（https://atom.io）。

有些人对代码编辑器比较挑剔。你不一定要使用我们推荐的编辑器，但我们建议不要使用 Vim、Vi 或 Emacs，除非你用过这些工具。一些编程的纯粹主义者只用这些工具来写代码（本书的一位作者就是这样），因为他们可以完全用键盘来操纵编辑器。但是，如果你没有任何经验就选择这些编辑器的话，可能会在阅读本书的过程中困难重重，因为你要同时学习两件事情。

一次只学习一件事情，多试用几款编辑器，直到找到可以让你轻松自如写代码的那一款。对于 Python 开发来说，最重要的是要有一款用起来舒服的编辑器，同时支持许多文件类型（能支持 Unicode 和 UTF-8）。

下载选定的编辑器并安装之后，运行程序检查是否安装成功。

1.2.6　安装IPython（可选）

如果你想安装更高级的 Python 解释器，我们推荐安装一个叫 IPython 的库（下载地址：http://ipython.org/install.html）。在附录 F 中我们讲了 IPython 的优点、用例，以及如何安装 IPython。IPython 并不是必需的，但它在 Python 入门阶段十分有用。

1.3　小结

本章我们学习了两个常见的 Python 版本。为了能够继续学习数据处理，我们还完成了一些初始安装工作：

(1) 我们安装并测试了 Python；
(2) 我们安装了 pip；
(3) 我们安装了代码编辑器。

想要入门 Python，这是最基本的安装。随着对 Python 和编程的深入学习，你会发现还有更复杂的安装与设置。现在我们的目标是让你尽快上手，而不是纠结于繁琐的安装过程。如果你想了解更高级的 Python 安装内容，请查阅附录 D。

在阅读本书的过程中，你可能会用到需要高级安装的工具。那时我们将教你如何从现有基本安装中创建更复杂的安装。现在，我们之前讲的安装内容已经可以让你迈出学习 Python 的第一步了。

恭喜，你已经完成了初始安装，还第一次运行了几行 Python 代码！在下一章里，我们将开始学习 Python 的基本概念。

第 2 章

Python基础

前面你已经在计算机上安装并运行了 Python，下面我们来学习一些 Python 的基础知识。后续章节的内容都建立在这些概念的基础上，我们需要首先掌握这些概念。

在上一章中，我们利用下面几行代码测试了 Python 的安装是否成功：

```
import sys
import pprint
pprint.pprint(sys.path)
```

读完本章，你将会明白上面每一行代码的含义，并学会相关术语，能够说明上述代码的作用。你还将学习 Python 的各种数据类型，对 Python 的入门概念有基本的了解。

我们会讲得比较快，把重点放在学习后续章节时需要知道的内容。后续章节中我们还会根据需要讲一些新概念。我们希望这种方法能让你将这些新概念应用于感兴趣的数据集和问题，并从中学到东西。

首先我们启动 Python 解释器。本章的 Python 代码都在解释器中运行。浏览一遍本章这样的介绍性内容是很容易的，但我们要着重强调动手敲代码的重要性。与学习一门口语类似，在实践中学习是最有效的学习方法。在你输入本书练习代码并运行的过程中，会遇到大量的错误，而调试过程（解决这些错误的过程）会让你学到很多知识。

启动 Python 解释器

在第 1 章里我们学过如何启动 Python 解释器。提醒一下，你首先需要打开命令行工具，然后输入 python（如果你安装了附录 F 中提到的 IPython，也可以输入 iPython）：

```
python
```

你应该会看到类似这样的输出（注意，提示符已经切换成 Python 解释器的提示符）：

```
Python 2.7.7 (default, Jun 2 2014, 18:55:26)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

如果没有特别说明的话，本章后面我们输入的所有内容都是在 Python 解释器中输入的。如果你用的是 IPython，提示符是类似 `In [1]:` 这样的。

2.1 基本数据类型

本节将学习 Python 中的简单数据类型。这些数据类型都在 Python 的信息处理过程中发挥重要作用。我们要学习的数据类型包括字符串型、整型、浮点型和其他非整数类型。

2.1.1 字符串

我们要学习的第一个数据类型是字符串型。你之前可能没有在这种语境下听过“字符串”这个词，但字符串实际上就是用引号标记的文本。字符串里可以包含数字、字母和符号。

下面都是字符串的例子：

```
'cat'
'This is a string.'
'5'
'walking'
'$GOObarBaz340   '
```

将上面任一字符串输入到 Python 解释器里，解释器会返回相同的内容。程序仿佛在说：“嘿，我听到你了。你说的是 `'cat'`（或任何你输入的内容）。”

只要字符串位于成对的引号（单引号或双引号都可以）之间，其内容无关紧要。字符串的首尾必须用相同的引号（单双均可）：

```
'cat'
"cat"
```

上面两个例子在 Python 中的含义是相同的。两种情况下，Python 都会返回单引号的 `'cat'`。有些人习惯于在代码中使用单引号，另一些人更喜欢用双引号。无论你用哪种引号，重要的是前后风格要保持一致。本书的两位作者更喜欢用单引号，因为输入双引号需要按住 Shift 键。输入单引号可以偷懒。

2.1.2 整数和浮点数

我们要学习的第二种和第三种数据类型是整型和浮点型，这是 Python 中处理数字的方法。首先来看整型。

1. 整数

你可能还记得数学课上学过的整数，但如果你忘记了，整数是指整数（an integer is a whole number）。下面是一些整数的例子：

```
10
1
0
-1
-10
```

将上面的整数输入到 Python 解释器里，解释器会返回相同的内容。

注意，上一节的字符串例子中有一个是 `'5'`。如果输入的数字是在引号中，Python 会将其看作字符串。在下面的例子中，第一个值和第二个值并不相等：

```
5
'5'
```

想测试二者是否相等，可以在解释器中输入：

```
5 == '5'
```

`==` 符号是用来测试两个值是否相等。测试的返回值是真（True）或假（False）。返回值是另一种 Python 数据类型，叫作布尔型。后面会讲到布尔型，这里先简单学习一下。布尔型可以判断语句是真还是假。在上面的语句中，我们问 Python：整数 `5` 与字符串 `'5'` 是否相等？ Python 返回了什么？怎样才能让语句的返回值为 `True` ？（提示：试一下两个都是整数或两个都是字符串的情况！）

你可能想知道，为什么有人会把数字当作字符串来存储。有时是因为使用不当——例如，代码将数字存储成 `'5'`，而实际上应该存储成 `5`，没有引号。再举一个例子，字段内容是人工填写的，里面可能既包含字符串又包含数字（例如，调查中人们可能输入“五”“5”或“V”）。它们都是数字，但是数字的不同表示方式。在这种情况下，你可能会先把它们存储成字符串，后面再进行处理。

将数字存储为字符串，最常见的原因之一是故意这么做，例如美国邮政编码的存储。美国的邮编共包含五位数字。在新英格兰地区和东北部的其他地方，邮编是以 0 开头的。在 Python 解释器中试着输入波士顿的一个邮编，一次作为字符串输入，一次作为整数输入。出现了什么情况？

```
'02108'
02108
```

在第二个例子中，Python 会抛出一个语法错误［`SyntaxError`，报错信息为无效的标记（`invalid token`），并且有一个箭头指向开头的 0］。在 Python 以及许多其他语言中，“标记”（token）是指特殊的单词、符号和标识符。在上面的例子中，Python 不知道如何处理以 0 开头的数（非八进制的数），这意味着它是一个无效的标记。

2. 浮点数、小数和其他非整数类型

Python 处理非整数的运算有许多种方法。如果你不了解每一种非整数数据类型的运算方

法，这些方法可能会令人感到困惑，而且似乎引入了舍入误差。

在 Python 中使用非整数时，Python 默认将其转换成浮点数。浮点数的表示方法采用的是各版本 Python 内置的浮点数据类型。也就是说，Python 存储的是该数值的近似值——仅反映特定精度水平的近似值。

在 Python 解释器中输入下面这两个数字，注意二者之间的不同：

```
2
2.0
```

第一个是整数。第二个是浮点数。我们做点计算，进一步了解这些数字的运算规则，以及 Python 如何对它们进行求值。在 Python 解释器中输入：

```
2/3
```

发生了什么？你得到的返回值是 `0`，但你可能本来以为会是 `0.6666666666666666`、`0.6666666666666667` 或类似的数字。问题在于，这些数字都是整型，整型无法完成分数运算。我们试着把其中一个数字变成浮点数：

```
2.0/3
```

现在我们得到了更精确的答案：`0.6666666666666666`。如果输入的数字中有一个是浮点数，计算的结果也是浮点数。

如前文所述，Python 中的浮点数可能会引起精度问题（https://docs.python.org/2/tutorial/floatingpoint.html）。浮点数的运算速度很快，但正是因为这一点，浮点数不够精确。

从原理上来说，Python 与你和计算器对待数字的方式都不同。在 Python 解释器中试试下面这两个例子：

```
0.3
0.1 + 0.2
```

对于第一行代码，Python 返回 `0.3`。对于第二行代码，你本来希望返回 `0.3`，但实际上你得到的返回值是 `0.30000000000000004`。`0.3` 和 `0.30000000000000004` 这两个值并不相等。如果对其中的细微差别感兴趣，可查阅 Python 文档（https://docs.python.org/2/tutorial/floatingpoint.html#tut-fp-issues）了解更多内容。

在本书中，需要考虑精度问题时，我们将使用 `decimal` 模块（或库，https://docs.python.org/2/library/decimal.html）。模块是可导入使用的代码段或代码库。`decimal` 模块可以让数字（整数或浮点数）的运算结果符合预期（与你在数学课上学到的概念一致）。

在下面的例子中，第一行代码从 `decimal` 模块中导入了 `getcontext` 和 `Decimal`，这样我们就可以在解释器环境中使用它们。接下来的两行代码利用 `getcontext` 和 `Decimal` 来做之前用浮点数做过的数学运算：

```
from decimal import getcontext, Decimal
getcontext().prec = 1
Decimal(0.1) + Decimal(0.2)
```

运行上面的代码，Python 会返回 `Decimal('0.3')`。现在你输入 `print Decimal(0.3)`，

Python 会返回 0.3，这也正是我们本来希望看到的值（而不是 0.30000000000000004）。

我们来一行一行地阅读这三行代码：

```
from decimal import getcontext, Decimal     ❶
getcontext().prec = 1                       ❷
Decimal(0.1) + Decimal(0.2)                 ❸
```

❶ 从 decimal 模块中导入 getcontext 和 Decimal。

❷ 将舍入精度设定为一位小数。decimal 模块将大部分的舍入和精度设置保存在默认的上下文（context）中。本行代码将上下文的精度改成只保留一位小数。

❸ 对两个小数（一个值为 0.1，一个值为 0.2）求和。

如果修改 getcontext().prec 的值会怎么样？动手试一下，然后重新运行最后一行代码。你应该会看到不一样的结果，这取决于你设置的是保留几位小数。

如前文所述，在数据处理过程中你会遇到许多数学上的细节问题。你可能需要完成的数学运算也有许多不同的方法，但在用到非整数时，使用小数型数据可以使计算精度更高。

> **Python 中的数字**
>
> Python 中的数字类型有不同的精度水平，这是 Python 语言的缺点之一。随着对数据处理的深入学习，我们将在本书中学到更多 Python 中与数字和数学相关的库。要是你现在就想知道，并且打算做一些不那么基础的数学运算，下面是一些你需要熟悉的 Python 库。
>
> - decimal（https://docs.python.org/2/library/decimal.html），用于定点运算和浮点运算。
> - math（https://docs.python.org/2/library/math.html），可以使用 C 语言标准所定义的数学函数。
> - numpy（http://docs.scipy.org/doc/numpy/reference/routines.math.html），Python 科学计算的基础包。
> - sympy（http://docs.sympy.org/latest/index.html），用于符号数学的 Python 库。
> - mpmath（http://mpmath.org/）用于任意精度实数和复数浮点运算的 Python 库。

我们已经学过了字符串型、整型和浮点型 / 小数型。下面以这些基本数据类型为基础，创建更复杂的数据类型。

2.2 数据容器

本节会讲到数据容器，里面装有许多数据点（data point）。但应该注意的是，这些容器本身也是 Python 的数据类型。Python 中有几种常见的容器：变量、列表和字典。

2.2.1 变量

变量为我们提供了保存数据的方法，这些数据包括字符串、数字或其他数据容器。变量由一串字符组成，通常是一个小写单词（或用下划线连接的多个单词），用来说明变量包含的内容。

我们试着创建一个简单的变量。在 Python 解释器中，试试下面的代码：

```
filename = 'budget.csv'
```

输入正确的话，解释器应该不会返回任何值。这与在 Python 解释器中输入字符串有所不同。如果在 Python 解释器中只输入 `'budget.csv'`，它会输出 `'budget.csv'`。

当你创建一个变量时，是将程序本来应该输出的数据赋值给这个变量。这也是创建新变量时没有返回值的原因。在上面的例子中，我们的变量叫作 `filename`，其中保存着我们输入的字符串（`'budget.csv'`），作为它的值。

面向对象编程

你可能听说过**面向对象编程**（object-oriented programming，OOP）。Python 是一门面向对象的编程语言。"面向对象编程"中的"对象"可以是本章我们学过的任意一种数据类型，例如字符串、变量、数字或浮点数。

在正文给出的例子中，我们的对象是一个字符串，此时保存在 `filename` 中。我们定义的每一个变量都是一个 Python 对象。在 Python 中，我们用对象来保存后面要用到的数据。这些对象通常具有不同的性质和行为，但它们都是对象。

例如，每一个整数对象都可以用 + 号（加法运算符）与另一个整数相加。继续学习 Python 的过程中，你会学到关于这些对象及其基础类型的性质和行为的更多内容，也会因此喜欢上面向对象编程的！

前面在创建一串字母并将其赋值给名为 `filename` 的变量时，我们遵循了变量命名的几条通用规则。不用担心记不住这些规则，但如果在代码中定义新变量时出现错误，一定要想起这些规则。

- 可以包含下划线，但不能包含连字符。
- 可以包含数字，但变量名不能以数字开头。
- 为了方便阅读，单词用小写字母，单词之间用下划线隔开。

试一试下面的代码：

```
1example = 'This is going to break.'
```

发生了什么？你得到了什么类型的错误？你应该得到的是语法错误，因为你违反了第二条规则。

只要不违反 Python 变量命名的规则，变量几乎可以任意命名。例如：

```
horriblevariablenamesarenotdescriptiveandtoolong = 'budget.csv'
```

可以看出，变量名太长，并且没有对变量内容进行说明。另外，缺少下划线，让人难以读懂。什么样的变量名是一个好的变量名？问自己一个问题：六个月之后，如果我已经把代码全部忘记了，什么样的变量名仍是有意义的，并能帮我理解代码的内容？

我们来看一个更合理的变量名——`cats`。如前一个例子所示，变量的值不一定要是文件名。

变量可以有许多不同的值和名字。我们假装在数猫咪的数量，所以要把一个整数赋值给变量 cats：

```
cats = 42
```

如果 Python 脚本记录了我们有多少只猫咪，我们不需要每时每刻都知道确切的数字。我们只需要知道这个值被保存在变量 cats 中，所以，当我们在解释器中调用 cats 时，或者在另一段代码中用到 cats 时，它总会返回当前猫咪的数量。

*调用*变量，是指请求 Python 返回该变量的值。我们来调用 cats。在解释器中输入 cats。你应该会得到 42 作为返回值。如果输入 filename，你应该会得到字符串 'budget.csv' 作为返回值。在计算机上试一下：

```
>>> cats
42
>>> filename
'budget.csv'
>>>
```

如果输入不存在的变量名（或者输错了变量名），你会得到以下错误：

```
>>> dogs
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'dogs' is not defined
```

前面说过，学会如何阅读错误信息是很重要的，这可以让你明白犯了什么错误，以及如何改正错误。在这个例子中，错误信息指出，*dogs* is not defined，意思是我们没有定义过一个叫作 dogs 的变量。由于我们没有定义过这个变量，所以 Python 不知道我们要调用的是什么。

在第一个例子中，如果你忘了在 'budgets.csv' 上加引号，也会得到相同的错误。在 Python 解释器中试一下：

```
filename = budget.csv
```

返回的错误信息是 NameError: *name 'budget'* is not defined。这是因为 Python 并不知道 budget.csv 应该是一个字符串。要记住，字符串总是用引号来标记。没有引号的话，Python 会试图将其解释成另一个变量。这个练习的要点就是，注意错误发生在哪一行，并问问自己：错误可能出在哪里？在 dogs 的例子中，错误信息指出，错误发生在第 1 行。如果有很多行代码，错误信息可能会指向第 87 行。

前面给出的所有例子都是短字符串或整数。变量也可以保存长字符串，甚至是跨很多行的字符串。在例子中我们选择用短字符串，是因为对你（或我们）来说，输入长字符串一点都不好玩。

试一下保存长字符串的变量。注意，字符串里含有单引号，我们必须用双引号来保存：

```
recipe = "A recipe isn't just a list of ingredients."
```

现在输入 recipe，你会得到保存的长字符串：

```
>>>recipe
"A recipe isn't just a list of ingredients."
```

变量的数据类型不一定要是字符串或整数。变量可以保存各种不同的 Python 数据类型，我们会在后续章节进一步学习这方面的内容。

2.2.2 列表

列表是具有某种共同关系的一组值。在 Python 中使用列表，和在日常语言中使用列表很相似。在 Python 中，将一列元素放到方括号（[]）中，元素之间用逗号隔开，即可创建列表。

我们在 Python 中创建一个食品的列表：

```
['milk', 'lettuce', 'eggs']
```

> 上面这个列表由字符串组成，不是由变量组成。这一点是可以看出来的，因为单词都用引号包裹。如果元素都是变量的话，是不会有两边的引号的。

按下 Return 键，Python 会返回以下内容：

```
['milk', 'lettuce', 'eggs']
```

你刚刚创建了第一个 Python 列表：字符串列表。你可以创建任意数据类型的列表，或者多种数据类型混合的列表（例如，同时包含浮点数和字符串的列表）。我们来创建一个同时包含整数和浮点数的列表：

```
[0, 1.0, 5, 10.0]
```

下面将我们的列表保存在变量中，这样就可以在后面的代码中调用它。变量很有用，因为变量可以让我们不必反复输入数据。如果列表很长，比如说有 5000 个元素那么长，手动输入数据很容易出错，而且效率不高。正如前文所述，变量是保存值的方法，将其保存在合理命名的容器中。

在 Python 解释器中尝试以下代码：

```
shopping_list = ['milk', 'lettuce', 'eggs']
```

按下 Return 键，你应该会看到新的一行。看起来似乎什么也没有发生。之前是将列表原样返回，还记得吗？现在 Python 将列表保存在 `shopping_list` 变量中。如果在 Python 提示符中输入 `shopping_list` 来调用变量，你应该会得到以下返回值：

```
shopping_list
['milk', 'lettuce', 'eggs']
```

列表也可以保存变量。比如说，我们有一个变量，里面保存的是动物收容所中动物的数量：

```
cats = 2
dogs = 5
horses = 1
```

现在可以将这些动物的数量放在一个列表中：

```
animal_counts = [cats, dogs, horses]
```

在 Python 解释器中输入 animal_counts，Python 会返回下面的值：

```
[2, 5, 1]
```

变量为我们保存信息。当我们输入变量名时，Python 返回的是变量中保存的值。

你还可以创建包含列表的列表。比如说，创建动物名字的列表：

```
cat_names = ['Walter', 'Ra']
dog_names = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
horse_names = ['Mr. Ed']
animal_names = [cat_names, dog_names, horse_names]
```

在 Python 解释器中输入 animal_names，Python 会返回下面的值：

```
[['Walter', 'Ra'], ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido'], ['Mr. Ed']]
```

你不必输入所有的名字来创建包含列表的列表。原始变量（cat_names，dog_names，horse_names）也是列表，它们仍然可用。例如，输入 cat_names，你会得到 ['Walter', 'Ra'] 作为返回值。

前面已经探索了列表，下面我们继续探索一个稍微复杂一点的数据容器，叫作字典。

2.2.3　字典

字典比变量或列表更加复杂。“字典”这个名字是很贴切的。将 Python 的字典看作传统意义上的字典——用来查找单词释义的工具。在 Python 的字典中，你要查找的单词叫作*键*（key），这些单词的释义叫作*值*（value）。在 Python 中，键对应一个值。

回头看动物的例子。animal_numbers 保存的是我们拥有的不同动物数量的列表，但我们不清楚哪个数字对应哪一种动物。字典是存储这种信息的好方法。

在下面的例子中，我们用动物种类作为键，每种动物的数量作为值：

```
animal_counts = {'cats': 2, 'dogs': 5, 'horses': 1}
```

如果想用键来访问一个值，可以从字典中访问键（类似在普通的字典中查单词）。为了在 Python 中实现这样的查找（比如查找我们拥有的狗狗的数量），可以输入以下代码：

```
animal_counts['dogs']
```

你应该看到的返回值是 5，因为我们在字典中（'dogs': 5）设置 'dogs' 键对应的值是 5。如你所见，想要存储匹配的键值对时，字典是很有用的。根据你的需求不同，字典可以是非常强大的，所以我们进一步研究同时使用列表和字典。

对于前面动物名字的列表，很难说清楚哪个名字列表属于哪一种动物。哪个列表里面是猫咪的名字，哪个列表里面是狗狗的名字，哪个列表里面是马的名字，谁也搞不清楚。但是，我们可以用字典把动物们区分清楚：

```
animal_names = {
    'cats': ['Walter', 'Ra'],
    'dogs': ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido'],
    'horses': ['Mr. Ed']
    }
```

存储同样的值还有另一种方法，它用到了更多的变量：

```
cat_names = ['Walter', 'Ra']                                   ❶
dog_names = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
horse_names = ['Mr. Ed']

animal_names = {
    'cats': cat_names,                                         ❷
    'dogs': dog_names,
    'horses': horse_names
    }
```

❶ 这一行代码将变量 `cat_names` 定义为猫咪名字的列表（字符串列表）。
❷ 这一行代码使用变量 `cat_names`，传递猫咪名字的列表作为字典中键 `'cats'` 对应的值。

虽然过程有所不同，但两种方法得到的是相同的字典[1]。随着对 Python 的深入学习，你将能够更好地判断，什么时候需要定义更多的变量，什么时候不这么做反而更好。现在你会发现，利用许多定义好的不同变量（例如 `cat_names` 和 `dog_names`）来创建新的变量（例如 `animal_names`），这是很容易的。

> 虽然 Python 有间距和格式的规则，但你的字典格式不一定非得和我们前面的一样。但是，你的代码应该尽可能易于阅读。保证代码的可读性，你自己以及其他合作的开发者都会因为这一点而感谢你的。

2.3 各种数据类型的用途

每种基本数据类型都可以做各种各样的事情。下面列出了我们目前学过的数据类型，并举例说明这些数据类型所能做的各种事情。

- 字符串
  - 大小写转换
  - 删除字符串末尾的空格
  - 分割字符串
- 整数和小数
  - 加减运算
  - 简单数学运算
- 列表
  - 在列表中增加或删除元素

注 1：两个字典并不完全相同，因为第二个字典中使用了可以被修改的对象。想详细了解二者的区别，请查阅附录 E。

  - 删除列表的最后一个元素
  - 列表重新排列
  - 列表排序
- 字典
  - 增加一个键 / 值对
  - 将指定的键设置为新的值
  - 利用键查找值

在上面这个列表中我们故意没有提到变量。变量能做的事情取决于它包含的数据。比如说，如果变量是一个字符串，那么它可以做所有字符串能做的事情。如果变量是一个列表，那么它可以做只有列表才能做的各种事情。

把数据类型看作名词，把它们能做的事情看作动词。在大多数情况下，数据类型能做的事情被称为方法（method）。想要访问数据类型的方法，或者说让数据类型做一些事情，你可以用点号（`.`）。比如说，如果你将一个字符串赋值给名为 `foo` 的变量，你可以输入 `foo.strip()` 来调用该字符串的 `strip` 方法。我们来看一下其中一些方法的作用。

在调用字符串的方法时，它们的行为是 Python 默认库的一部分（类似于手机上预装的默认应用），所有 Python 版本共享这一默认库。在每一台运行 Python 的计算机上都可以调用这些方法，因此每一个 Python 字符串都可以共享相同的方法（就像每一台手机都可以打电话，每一台苹果手机都可以发送 iMessage 一样）。Python 标准库（也叫作 stdlib，https://docs.python.org/2/library/）中包含了大量的内置方法和基本数据类型，其中包括你正在使用的 Python 数据类型。

2.3.1 字符串方法：字符串能做什么

回头看最开始定义的变量 `filename`。我们开始时用 `filename = 'budget.csv'` 来定义变量。这种定义方法很方便。但有些时候，事情并没有那么简单。来看几个例子：

```
filename = 'budget.csv         '
```

注意，现在 `filiname` 字符串里有许多多余的空格，可能需要将其删掉。可以用 Python 字符串的 `strip` 方法，这是一个内置函数，可以从头到尾删除字符串中多余的空格：

```
filename = 'budget.csv         '
filename = filename.strip()
```

如果你没有对变量重新赋值的话（令 `filename` 等于 `filename.strip()` 的输出值），你对 `filename` 所做的修改不会被保存。

在 Python 解释器中输入 filename，现在你应该可以看到，空格已经被删除。

比如说，文件名需要全部用大写字母。可以用 Python 字符串内置的 upper 函数将所有的字母转换成大写：

```
filename = 'budget.csv'
filename.upper()
```

从输出应该可以看出，现在文件名已经全部大写了：

```
'BUDGET.CSV'
```

在这个例子中，我们没有将大写的字符串重新赋值给变量 filename。在解释器中再次调用 filename 时会发生什么？输出应该还是 'budget.csv'。如果你不希望修改你的变量，只想对变量进行转换后使用一次，可以调用类似 upper 这样的方法，这些方法会返回修改后的字符串，但不会改变变量本身。

如果想对变量重新赋值，用同一个变量名保存返回值，应该怎么办呢？接下来，将变量 filename 的值改成全部大写：

```
filename = 'budget.csv'          ❶
filename = filename.upper()      ❷
```

❶ 在本行之后调用 filename，输出是 'budget.csv'。
❷ 在本行之后调用 filename，输出是 'BUDGET.CSV'。

可以将代码压缩成一行来运行：

```
filename = 'budget.csv'.upper()
```

代码的行数有时是你个人的风格或偏好。你可以随意选择你认为合理的方式，但要保持代码清晰、易读、明了。

在上面这些例子中只讲了两个字符串方法：strip 和 upper，但还有许多其他的内置字符串方法。随着在数据处理过程中遇到更多的字符串，我们会学习更多的字符串方法。

2.3.2　数值方法：数字能做什么

整数和浮点数 / 小数都是数学对象。如果你输入 40 + 2，Python 会返回 42。如果你想把结果保存在变量中，可以将其赋值给一个变量，正如我们在字符串的例子中所做的那样：

```
answer = 40 + 2
```

现在你输入 answer，得到的返回值是 42。整数能做的大部分事情都是可以预知的，但有可能你需要使用一些特殊的格式，这样 Python 解释器才能理解你想做的数学运算。例如，如果你想计算 42 的平方，那你需要输入 42**2。

整数、浮点数和小数也还有许多其他的方法，我们会在学习数据处理的过程中遇到其中一些方法。

加法和减法

对于 Python 的其他数据类型，你也可以做加法，例如字符串和列表。试一试下面的代码：

```
'This is ' + 'awesome.'
```

和

```
['Joker', 'Simon', 'Ellie'] + ['Lishka', 'Turtle']
```

试试用减法，看看会发生什么？运行下面的代码会报错，你能从中学到什么？

```
['Joker', 'Simon', 'Ellie', 'Lishka', 'Turtle'] - ['Turtle']
```

你应该会得到报错信息：`TypeError: unsupported operand type(s) for -: 'list' and 'list'`。这告诉我们，Python 列表支持加法，却不支持减法。这是因为 Python 开发人员对每种数据类型支持的方法所做的选择。如果你想了解如何对列表做减法，可以查看 Python 官网上列表的 `remove` 方法（https://docs.python.org/2/tutorial/datastructures.html）。

2.3.3 列表方法：列表能做什么

列表有几个必须知道的方法。我们从一个空列表开始，用一个方法来添加元素。

首先，像这样定义一个空列表：

```
dog_names = []
```

在解释器中输入 `dog_names`，返回的是 `[]`，这是 Python 显示空列表的方式。在本章前面的内容里，这个变量中保存了好几个名字，但在上一行代码中我们重新定义了这个变量，现在它是一个空列表。内置的 `append` 方法可以向列表中添加元素。现在利用这一方法将“Joker”添加到列表中：

```
dog_names.append('Joker')
```

现在输入 `dog_names`，返回的列表中将包含一个元素：`['Joker']`。

下面你自己来操作，利用 `append` 方法创建这样的列表：

```
['Joker', 'Simon', 'Ellie', 'Lishka', 'Turtle']
```

比如说，你不小心添加了一个猫咪的名字，`'Walter'`：

```
dog_names.append('Walter')
```

你可以利用 Python 列表内置的 `remove` 方法删除这个元素：

```
dog_names.remove('Walter')
```

列表还有许多其他的内置方法，但 `append` 和 `remove` 是其中最常用的两个。

2.3.4 字典方法：字典能做什么

为了学习一些有用的字典方法，我们来从头创建一个动物数量的字典。

在下面的例子中，我们创建了一个空的字典。然后添加了一个键，并给定了这个键对应的值：

```
animal_counts = {}
animal_counts['horses'] = 1
```

向字典添加元素（`animal_counts['horses']`）与向列表添加元素略有不同。这是因为字典既有键又有值。在上面的例子中，键是 `'horses'`，值是 `1`。

我们用动物数量来定义字典的其他部分：

```
animal_counts['cats'] = 2
animal_counts['dogs'] = 5
animal_counts['snakes'] = 0
```

现在在 Python 解释器中输入 `animal_counts`，你应该会得到以下字典：`{'horses': 1, 'cats': 2, 'dogs': 5, 'snakes': 0}`。（Python 字典不会保存元素的顺序，所以你看到的输出可能会有所不同，但应该包含相同的键值对）。

这里讲的是一个非常小的例子，但是编程并非总是这样方便。试想由世界上所有家畜的数量构成的字典。作为程序员，我们可能不知道这个 `animal_counts` 字典中包含的所有动物种类。在处理一无所知的大型字典时，我们可以利用字典方法来了解字典的更多内容。下面这个命令返回字典包含的所有键：

```
animal_counts.keys()
```

如果你把前面的练习都做了，那么在解释器中输入上面的代码会返回一个键的列表，如下所示：

```
['horses', 'cats', 'dogs', 'snakes']
```

对于上面任何一个键，你可以从字典中检索到与其对应的值。下面的查询将返回狗狗的数量：

```
animal_counts['dogs']
```

这行代码的输出是 `5`。

如果你愿意，可以将这个值保存在一个新变量中，这样你就无需再次查询：

```
dogs = animal_counts['dogs']
```

现在，你直接输入变量 `dogs`，Python 将返回 `5`。

这就是你可以对字典做的一些基本操作。随着用代码去解决越来越复杂的问题，我们也会更深入地学习字典，正如字符串和列表一样。

2.4 有用的工具：type、dir和help

Python 标准库中有几个内置工具，可以帮你确定变量的数据类型或对象类型，并给出这些变量能做的事情（即它们都有哪些方法）。本节将学习三个工具，它们都是 Python 标准库的一部分。

2.4.1 type

type 可以帮你确定你的对象属于哪种数据类型。想在 Python 代码中做到这一点，将变量放到 type() 的括号里，例如，如果变量名是 dogs，那就在 Python 提示符后输入 type(dogs)。当你用变量保存数据，并想知道变量里的数据是什么类型时，这一方法是非常有用的。回忆本章前面邮政编码的例子。

对于值 20011，这里有两种不同的用法。在第一个例子中，它是保存成字符串的邮编。在第二个例子中，它是一个整数：

```
'20011'
20011
```

如果将这两个值保存在变量中，将更难以确定变量的类型，我们可能不知道或不记得用的是字符串还是整数。

如果将值传递给内置方法 type，Python 就会告诉我们对象属于那种数据类型。试一下：

```
type('20011')
type(20011)
```

第一行返回的是 str，第二行返回的是 int。将列表传递给 type 会返回什么？变量呢？

在你试图排查错误时，或者运行其他人的代码时，确定对象的类型是很有用的。还记得我们试图从一个列表中减去另一个列表吗（见 2.3.2 节）？好吧，你也不能从一个字符串中减去另一个字符串。所以，与整数 20011 相比，字符串 '20011' 具有许多不同的方法以及用例。

2.4.2 dir

dir 会返回一个内置方法与属性的列表，帮你列出特定数据类型能做的所有事情。我们用字符串 'cat,dog,horse' 来试一下：

```
dir('cat,dog,horse')
```

暂时忽略返回的列表中开头的那些项（以双下划线开头的那些字符串）。这些是 Python 使用的内部方法或*私有方法*。

最有用的方法包含在返回列表的第二部分。许多方法的用途显而易见，或者说是自说明的（self-documenting）。你应该会看到本章前面使用的一些字符串方法：

```
[...,
'__sizeof__',
'__str__',
'__subclasshook__',
'_formatter_field_name_split',
'_formatter_parser',
'capitalize',
'center',
'count',
'decode',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'index',
'isalnum',
'isalpha',
'isdigit',
'islower',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'zfill']
```

看一下字符串 `'cat,dog,horse'`，它看起来像是保存在字符串里的列表。实际上它是单一值，但利用 Python 字符串内置的 `split` 方法，我们可以以逗号为分隔符，将字符串切分成更小的字符串，像这样：

```
'cat,dog,horse'.split(',')
```

Python 将返回一个列表：

```
['cat', 'dog', 'horse']
```

下面对这个列表调用 `dir` 函数：

```
dir(['cat', 'dog', 'horse'])
```

和字符串相比，列表可选的方法没有那么多，但我们来尝试其中几个。首先，将列表转化为变量。你现在应该知道如何将列表赋值给变量，这是一个例子：

```
animals = ['cat', 'dog', 'horse']
```

前面用 `dir` 作用在列表上给出了许多新方法，现在在变量 `animals` 上试用其中一些方法：

```
animals.reverse()
animals.sort()
```

运行每一行代码后，打印输出 `animals` 的值，这样你就能看出这些方法是如何改变列表的。你预期的输出是什么样的？它与你看到的相同吗？尝试将 `dir` 方法作用在整数和浮点数上。（提示：`dir` 只能传入一个对象，所以试着输入 `dir(1)` 或 `dir(3.0)`）。其中有没有你没有想到的方法？

如你所见，`dir` 可以让你深入了解每一种 Python 数据类型的内置方法。在利用 Python 进行数据处理时，这些方法是很有价值的。建议你花时间对感兴趣的上表列出的方法都尝试一下，并用不同的数据类型测试更多的方法。

2.4.3 help

本章要学习的第三个有用的 Python 内置方法是 `help` 方法。这一方法会返回对象、方法或模块的文档——虽然经常以技术性很强（有时很难懂）的文字书写。来看一下 `split` 方法的帮助文档，这是我们在前一节用过的方法。如果你不知道需要将字符串的分隔符放在括号内，怎么能知道如何使用 Python 字符串的 `split` 方法呢？假设我们不知道如何使用 `split`，不传入 `','` 来调用这一方法：

```
animals = 'cat,dog,horse'
animals.split()
```

代码的返回值如下：

```
['cat,dog,horse']
```

看起来不错，对吧？但经不起仔细观察。正如我们所见，Python 将字符串转换成一个列表，但并没有利用逗号对多个单词进行分割。这是因为内置的 `split` 方法默认按空格分割字符串，不是按逗号分割。我们需要在方法中传入一个逗号字符串（`','`），来告诉 Python 按逗号对字符串进行分割。

为了帮助理解这个方法是如何工作的，我们将其传递给 `help`。前面把变量 `animals` 变成了列表，所以首先我们必须重新定义这个变量。我们把它变回字符串，然后查看 `split` 的工作原理：

```
animals = 'cat,dog,horse'
help(animals.split) ❶
```

❶ 本行代码将 animals.split（没有 ()）传递给 help 方法。你可以向 help 方法中传入任何对象、方法或模块，但如前所见，在传入方法时不应该把尾部的括号也包括进去。

Python 的返回值如下：

```
split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are removed
    from the result.
```

帮助文档的第一行说的是：S.split([sep [,maxsplit]]) → list of strings。翻译成汉语，这告诉我们，对于字符串（S），我们有一个方法（split）、第一个可选参数（也就是可以传入的对象）sep 以及第二个可选参数 maxsplit。参数名两边的方括号（[]）表明它们是可选的，不是必需的。这个方法返回（->）一个字符串列表。

下一行说的是："Return a list of the words in the string S, using sep as the delimiter string." sep 是被传入 split 方法的参数，作用是分隔符（seperator）。分隔符（delimiter）是用来分割字段的单个字符或一串字符。例如，在一个逗号分隔文件中，逗号就是分隔符。逗号还是我们所创建字符串的分隔符，因为它把我们希望出现在列表中的单词分隔开。

阅读完帮助文档后（利用方向键上下翻页），你可以输入 q 退出 help。

help 文档还告诉我们，如果没有指定其他分隔符，默认的分隔符是空白。这告诉我们，如果我们有一个字符串 'cat dog horse'，split 方法不需要我们在 () 内传入分隔符。如你所见，内置的 help 函数可以告诉你许多关于如何使用某个方法的内容，还可以告诉你这个方法是否适合你正在解决的问题。

2.5　综合运用

我们利用刚学到的新技术来做一个测验。试着完成以下内容。

(1) 创建一个字符串、一个列表和一个字典。
(2) 利用 dir 方法查找每种数据类型可用的方法。
(3) 试用一些你发现的内置方法，直到某个方法抛出了错误。
(4) 利用 help 查看该方法的文档。试着理解这个方法是做什么的，并试着搞清楚，为了让这个方法正常运行，你可能还需要做些什么。

恭喜！你刚刚学会了如何编程。编程不是死记硬背，相反，编程是在出错时排查并解决错误。

2.6 代码的含义

在本章开头我们承诺过，在本章结束时你会理解这三行代码：

```
import sys
import pprint
pprint.pprint(sys.path)
```

根据目前所学的内容，我们把这三行代码分开来看。在 2.1.2 节中，我们导入了 decimal 库。看起来我们正从 Python 标准库中导入两个模块——sys 和 pprint。

看一下这些模块的帮助文档（一定要确定你已经导入了这些模块，否则 help 会抛出错误！）。由于 pprint 更容易读懂，我们先来看它的帮助文档：

```
>>>import pprint
>>>help(pprint.pprint)

Help on function pprint in module pprint:

pprint(object, stream=None, indent=1, width=80, depth=None)
    Pretty-print a Python object to a stream [default is sys.stdout].
```

很好。根据 pprint.pprint() 的文档，这个方法将传入的内容以易读的形式显示出来。

在上一章中我们学过，sys.path 给出 Python 寻找模块的位置。sys.path 的数据类型是什么？

```
import sys
type(sys.path)
```

是列表。我们知道列表怎么用！现在我们还知道，将一个列表传入 pprint.pprint，列表的输出格式会非常美观。我们把它用在包含列表的列表上，里面保存的是动物名字。首先，再多加一些名字，使列表变得很乱：

```
animal_names = [
    ['Walter', 'Ra', 'Fluffy', 'Killer'],
    ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido'],
    ['Mr. Ed', 'Peter', 'Rocket','Star']
    ]
```

下面将 pprint 作用于变量 animal_names:

```
pprint.pprint(animal_names)
```

得到的返回值如下：

```
[['Walter', 'Ra', 'Fluffy', 'Killer'],
 ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido'],
 ['Mr. Ed', 'Peter', 'Rocket', 'Star']]
```

总结一下，这是最开始这三行代码每一行的作用：

```
import sys      ❶
import pprint   ❷
```

```
pprint.pprint(sys.path) ❸
```

❶ 导入 Python 的 sys 模块。
❷ 导入 Python 的 pprint 模块。
❸ 将列表 sys.path 传递给 pprint.pprint，将列表清晰易读地显示出来。

如果将字典传入 pprint.pprint 会怎么样？你应该会看到格式优美的字典输出。

2.7 小结

数据类型和容器是 Python 理解并存储数据的方式。数据类型有许多种，本章只学习了其中重要的几种，如表 2-1 所示。

表2-1：数据类型

名称	举例
字符串	'Joker'
整数	2
浮点数	2.0
变量	animal_names
列表	['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
字典	{'cats': 2, 'dogs': 5, 'horses': 1, 'snakes': 0}

如你所知，有些数据类型可以包含在其他数据类型之中。列表可以是许多字符串或整数，或二者都有。变量可以是列表、字典、字符串或小数。看一下变量 animal_names，列表也可以是包含列表的列表。随着我们学习更多的 Python 知识，也会更深入地学习这些数据类型、它们的工作原理，以及如何利用它们来满足我们数据处理的需求。

本章我们还学习了 Python 的内置方法，以及能用对象所做的事情。另外，我们学习了几个简单的 Python 方法和工具。利用这些方法和工具，我们可以判断对象的数据类型及其用途。表 2-2 对这些工具做了总结。

表2-2：辅助工具

举例	用途
type('Joker')	返回 'Joker' 的对象类型
dir('Joker')	返回一个列表，给出对象 'Joker' 可以做的所有事情（方法和属性）
help('Joker'.strip)	返回给定方法（在本例中是 strip）的说明文档，以便我们更好地了解如何使用它

下一章我们将学习如何打开各种文件类型，以及如何将数据存储成本章学过的 Python 数据类型。通过将文件中的数据转换成 Python 对象，我们可以充分发挥 Python 的威力，数据处理将很快变成一件容易的事情。

第 3 章

供机器读取的数据

数据可以存储成许多不同的格式和文件类型。某些格式存储的数据很容易被机器处理，而另一些格式存储的数据则容易被人工读取。微软的 Word 文档属于后者，而 CSV、JSON 和 XML 文件则属于前者。本章我们将学习如何读取那些容易被机器处理的文件，在第 4 章和第 5 章我们将讨论那些供人工读取的文件。

> 以易于机器理解的方式来存储数据的文件格式，通常被称作机器可读的（machine readable）。常见的机器可读格式包括：
>
> - 逗号分隔值（Comma-Separated Values，CSV）
> - JavaScript 对象符号（JavaScript Object Notation，JSON）
> - 可扩展标记语言（eXtensible Markup Language，XML）
>
> 在口语和书面语中，提到这些数据格式时通常使用它们的短名字（如 CSV）。我们将使用这些缩写。

在寻找数据、向组织或机构发出数据请求时，你能找到最好的资源就是本章讲到的这些格式。与易于人工读取的格式相比，这些格式更容易被 Python 脚本处理，在数据网站上通常也很容易找到。

创建代码主文件夹

为了能够顺利完成本章的例子和代码，你需要将文件保存到本地计算机。你应该创建一个文件夹（如果之前还没有创建的话），用来保存 Python 代码和数据文件。文件夹的名字要直观，比如叫 data_wrangling（数据处理）。然后在这个文件夹中创建一个子文件夹，用来保存与本书相关的代码 [比如叫 code（代码）]。这有助于保持你的文件夹结构清晰，命名直观。

如果你按照上面的提示操作，应该创建好了一个像这样的文件夹：~/Projects/data_wrangling/code。

在基于 Unix 的系统中（Linux 和 Mac），~ 符号代表主目录，用命令行访问比较方便。

在 Windows 系统中，主目录位于 Users 文件夹下，所以你的文件夹位置是在 C:\Users\<your_name>\Projects\data_wrangling。

在本书的数据仓库中（https://github.com/jackiekazil/data-wrangling）可以下载代码示例，并将其移动到你的项目文件夹中。在阅读本章的过程中，我们假定，从上述仓库下载的数据与你编写的 Python 代码位于同一文件夹下。这样我们就不必担心文件定位问题，可以专心研究用 Python 导入数据。

3.1 CSV数据

我们要学习的第一个机器可读的文件格式是 CSV。CSV 文件（简称为 CSV）是指将数据列用逗号分隔的文件。文件的扩展名是 .csv。

另一种数据类型，叫作制表符分隔值（tab-separated values，TSV）数据，有时也与 CSV 归为一类。TSV 与 CSV 唯一的不同之处在于，数据列之间的分隔符是制表符（tab），而不是逗号。文件的扩展名通常是 .tsv，但有时也用 .csv 作为扩展名。从本质上来看，.tsv 文件与 .csv 文件在 Python 中的作用是相同的。

如果文件的扩展名是 .tsv，那么里面包含的很可能是 TSV 数据。如果文件的扩展名是 .csv，那么里面包含的可能是 CSV 数据，但也可能是 TSV 数据。一定要打开文件查看一下，这样你可以在导入数据之前就明确所处理的数据类型。

对于本章的 CSV 实例，我们采用的是来自世界卫生组织（WHO）的数据。WHO 拥有许多大型数据集（http://apps.who.int/gho/data/node.main），并且提供多种格式的数据。本例选择的数据集包含全球各国的预期寿命。访问网页（http://apps.who.int/gho/data/node.main.3?lang=en）查看预期寿命数据，你会发现这个数据集有几个不同的版本。本例中使用的是 CSV（纯文本，下载地址：http://apps.who.int/gho/athena/data/data-text.csv?target=GHO/WHOSIS_000002,WHOSIS_000001,WHOSIS_000015&profile=text&filter=COUNTRY:*;REGION:AFR;REGION:AMR;REGION:SEAR;REGION:EUR;REGION:EMR;REGION:WPR;SEX:*）。

用文本编辑器[1]打开这个 CSV 文件，你会看到类似表 3-1 的许多行数据。

注 1：为了完成本章练习，你需要一个好用的文本编辑器。如果还没有安装的话，你可以按 1.2.5 节的说明来安装。

表3-1：两个样本数据记录[a]

CSV标题	样本记录1	样本记录2
指标	60 岁时预期寿命（年）	出生时预期寿命（年）
发布状态	已发布	已发布
年份	1990	1990
WHO 地区	欧洲	美洲
世界银行收入分组	高收入	中低收入
国家	捷克共和国	伯利兹
性别	女性	男女合计
示值	19	71
数值大小	19.00000	71.00000
最低值	无	无
最高值	无	无
备注	无	无

a 加粗项包含在下文的样本数据中。

为了让数据更容易阅读，下面给出一个数据样本，其中只包含经过挑选的特定字段（表 3-1 中的加粗项）。在文本编辑器中打开 CSV 文件，你看到的数据应该与其类似：

```
"Year","Country","Sex","Display Value","Numeric"
"1990","Andorra","Both sexes","77","77.00000"
"2000","Andorra","Both sexes","80","80.00000"
"2012","Andorra","Female","28","28.00000"
"2000","Andorra","Both sexes","23","23.00000"
"2012","United Arab Emirates","Female","78","78.00000"
"2000","Antigua and Barbuda","Male","72","72.00000"
"1990","Antigua and Barbuda","Male","17","17.00000"
"2012","Antigua and Barbuda","Both sexes","22","22.00000"
"2012","Australia","Male","81","81.00000"
```

预览 CSV 文件的另一种方法是用电子表格程序打开，比如 Excel 或 Google Spreadsheets。这些程序将每一个数据条目显示为单独的一行。

3.1.1　如何导入CSV数据

前面我们学习了一点关于数据的知识，下面用 Python 打开这个文件，并将数据转换成 Python 可以理解的格式。这只要几行代码：

```
import csv

csvfile = open('data-text.csv', 'rb')
reader = csv.reader(csvfile)

for row in reader:
    print row
```

我们来一行一行地阅读上面的代码。在上一章里，我们所有的代码都是在 Python 解释器中

输入的，但随着代码变得越来越长、越来越复杂，在文件中编写代码并运行要方便一些。读完这段代码，我们将把它保存在一个 .py 文件中（.py 文件是一个 Python 文件），然后在命令行中运行这个文件。

脚本的第一行代码导入了一个叫作 `csv` 的库：

```
import csv
```

Python 库是一个代码包，提供了你可以在 Python 程序中使用的功能。这里导入的 `csv` 库是 Python 标准库（或 stdlib）的一部分，随 Python 一起安装。将库导入到文件中后，我们就可以使用这个库。如果没有这个库的话，脚本将会变得很长——`csv` 库提供了辅助函数，这样一来，为了完成更复杂的任务，我们就不必写这么多的代码。

第二行代码将 data-text.csv 文件传入 `open` 函数，这个文件应该和脚本位于同一文件夹下：

```
csvfile = open('data-text.csv', 'rb')
```

> *函数*（function）是一段代码，在被调用时执行相应的任务。它和我们在第 2 章学过的 Python 数据类型的*方法*十分相似。函数有时会接收一个（或多个）输入。这些输入叫作*参数*（argument）。函数的功能是基于参数的。函数有时也会返回一个输出，可以被保存或使用。

`open` 是 Python 的内置函数（这里列出了 Python 所有的内置函数：https://docs.python.org/2/library/functions.html），也就是说，打开文件的行为是如此常见，所以 Python 核心贡献者认为应把这个行为添加到 Python 的默认安装里。在使用 `open` 函数时，我们传入一个文件名作为第一个参数（这里用的是 `'data-text.csv'`），然后选择指定文件打开的模式（我们用的是 `'rb'`）。如果查看 `open` 函数的文档（https://docs.python.org/2/library/functions.html#open），你会发现，`'rb'` 参数的意思是我们以*只读方式*和*二进制方式*打开文件。以二进制方式打开文件，可以让代码在 Windows 上和基于 Unix 的操作系统上都能运行。另一种常见的模式是*写入*（`'w'` 或 `'wb'`，后者表示以*二进制*方式写入）。

> 如果你想读取文件，以只读方式打开文件。如果你想写入文件，以写入方式打开文件。

我们将这个函数的输出保存在变量 `csvfile` 中。现在 `csvfile` 保存一个打开的文件作为它的值。

在下一行代码中，我们将 `csvfile` 传递给 `csv` 模块的 `reader` 函数。这个函数的作用是让 `csv` 模块将打开的文件当作 CSV 来读取：

```
reader = csv.reader(csvfile)
```

函数 `csv.reader(csvfile)` 的输出保存在变量 `reader` 中。现在 `reader` 变量保存的是已打开文件的 Python CSV reader。有了这个 CSV reader，我们用简单的 Python 命令就可以轻松查看文件中的数据。最后一段代码中，我们使用了所谓的 `for` 循环。

`for` 循环是一种遍历 Python 对象的方法，通常与列表一起使用。`for` 循环告诉 Python 代

码：“对于这个列表中的每一个元素，做点什么。”在一个 for 循环中，for 后面的第一个单词是用来保存列表（或其他可迭代对象）中每一个对象的变量。for 循环下面的代码利用这个变量对该元素执行更多的操作或计算。因此，最好用有意义的单词做变量名，这样你和其他人都能轻松阅读并理解代码。

> 还记得第 2 章里出现过的“无效的标记”（invalid token）吗？ for 是 Python 里另一个特殊的标记（token），只能用来创建 for 循环。标记可以将我们在解释器或脚本中输入的内容翻译成计算机能够执行的命令。

试着在 Python 解释器中运行下面的代码：

```
dogs = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
for dog in dogs:
    print dog
```

利用这个 for 循环，我们将每一只狗狗的名字都保存在 for 循环的 dog 变量中。对于 for 循环的每一次迭代，我们打印出狗狗的名字（保存在变量 dog 中）。当程序遍历完每一只狗狗的名字（或列表中的每一个元素）后，代码停止运行。

在 IPython 中退出缩进代码块

在 IPython 终端里编写 for 循环或其他缩进代码块时，一定要检查一下，确保提示符从缩进代码块的样式 ... 变成了一个新的 In 提示符。最简单的方法就是，在完成最后一行缩进代码后敲下 Return 键。你应该看到一个新的 In 提示符，然后再输入循环之外的代码：

```
In [1]: dogs = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']

In [2]: for dog in dogs:
   ...:     print dog    ❶
   ...:                  ❷
Joker
Simon
Ellie
Lishka
Fido

In [3]:                  ❸
```

❶ IPython 的自动缩进提示符（...: 后面跟着四个空格）。

❷ 在空行按下 Return 键，退出缩进代码块并运行代码。

❸ IPython 的代码运行结束后，出现了新的提示符。

在我们用来读取 CSV 的代码中，reader 对象是一个保存数据行的 Python 容器。在 reader 的 for 循环中，我们将每一行数据保存在变量 row 中。下一行代码的意思是，我们让 Python 打印出每一行数据：

```
for row in reader:
    print row
```

现在我们已经能够导入数据并对数据进行遍历，下面开始对数据进行真正的探索。

3.1.2　将代码保存到文件中并在命令行中运行

作为开发者，在写代码时，即使是中间过程的部分代码片段，你也会希望保存下来，以便后续检查和使用。保持代码结构清晰，并及时保存代码，即使中途被打断，你也可以从上次中断的地方顺畅地继续工作。

我们将到目前为止所有的代码保存到文件中并运行。代码应该是这样的（如果你还没有完成这一步的话，打开文本编辑器，创建一个新文件，将这段代码输入进去）：

```
import csv

csvfile = open('data-text.csv', 'rb')
reader = csv.reader(csvfile)

for row in reader:
    print row
```

> 注意大小写、间距和换行。如果各行的代码间距各不相同或者大小写错误的话，代码是无法正常运行的。一定要严格按上面的代码输入，利用四个空格来缩进。这一点很重要，因为 Python 区分大小写，并利用缩进来表示代码的结构。

用文本编辑器将代码保存为 .py（Python）文件。完整的文件名应该是像这样的：import_csv_data.py。

将数据文件 data-text.csv 放到你刚刚保存 Python 文件的同一个文件夹内。如果你想把文件放到其他位置，需要对文件位置对应的代码做适当修改。

打开不同位置的文件

在目前的代码中，我们将文件路径传入 open 函数，像这样：

```
open('data-text.csv', 'rb')
```

但如果数据文件位于一个叫作 data 的子文件夹，我们需要修改脚本，让它去那里寻找数据文件。修改后的代码如下：

```
open('data/data-text.csv', 'rb')
```

在上面的例子中，我们的文件结构是像这样的：

```
data_wrangling/
`-- code/
     |-- import_csv_data.py
     `-- data/
          `-- data-text.csv
```

如果你找不到自己的文件，可以在 Mac 或 Linux 计算机上打开命令行，使用下列命令来浏览文件夹。

- `ls` 返回文件的列表。
- `pwd` 给出当前位置。
- `cd ../` 进入上层文件夹。
- `cd ../../` 向上移动两层目录。
- `cd data` 进入 data 文件夹，该文件夹位于当前文件夹下（可以用 `ls` 来查看）。

查阅附录 C 可以了解用命令行在文件夹之间跳转的更多内容，其中还包括针对 Windows 用户的整整一节内容。

保存好文件后，你可以用命令行来运行它。打开命令行（终端或 cmd），跳转到文件所在的位置。假设你把文件放在 ~/Projects/data_wrangling/code 中。想要在 Mac 命令行中跳转到那里，你可以使用变更目录或文件夹命令（`cd`）：

```
cd ~/Projects/data_wrangling/code
```

跳转到正确的位置后，你就可以运行 Python 文件。到目前为止，我们都是在 Python 解释器中运行代码。我们将文件保存为 import_csv_data.py。想要在命令行中运行 Python 文件，你只需要输入 `python`，敲空格，然后输入文件名即可。我们来试一下运行 Python 文件：

```
python import_csv_data.py
```

你得到的输出看起来应该像一串列表——和下面给出的数据类似，但数据量要大得多。

```
['Healthy life expectancy (HALE) at birth (years)', 'Published', '2012',
 'Western Pacific', 'Lower-middle-income', 'Samoa', 'Female', '66',
 '66.00000', '', '', '']
['Healthy life expectancy (HALE) at birth (years)', 'Published', '2012',
 'Eastern Mediterranean', 'Low-income', 'Yemen', 'Both sexes', '54',
 '54.00000', '', '', '']
['Healthy life expectancy (HALE) at birth (years)', 'Published', '2000',
 'Africa', 'Upper-middle-income', 'South Africa', 'Male', '49', '49.00000',
 '', '', '']
['Healthy life expectancy (HALE) at birth (years)', 'Published', '2000',
 'Africa', 'Low-income', 'Zambia', 'Both sexes', '36', '36.00000', '', '', '']
['Healthy life expectancy (HALE) at birth (years)', 'Published', '2012',
 'Africa', 'Low-income', 'Zimbabwe', 'Female', '51', '51.00000', '', '', '']
```

你得到上面的输出了吗？如果没有的话，花一分钟的时间阅读你得到的错误信息。从中发现出错的地方可能在哪里了吗？花点时间去搜索错误的原因，看看其他人解决相同的错误都用了哪些方法。关于如何解决错误，如果你需要额外的帮助，可查阅附录 E。

从现在开始，我们将会在代码编辑器里编写大多数代码，保存文件，然后在命令行中运行。Python 解释器仍然是一个有用的工具，可以用来测试代码片段，但随着代码变得越来越长、越来越复杂，在代码提示符中维护代码将越来越困难。

无论是我们正在写的代码，还是我们将面对的许多其他问题，解决问题的方法往往不止一种。`csv.reader()` 返回的是一个数据的列表，里面包含的是文件中的每一行数据，在我们刚开始处理问题时，这是一个很容易理解的方法。下面要对脚本做少许修改，将列表行改成字典行。这样在我们探索数据集的过程中，读取数据、对比数据和理解数据会变得更加容易。

在文本编辑器中，将第 4 行 `reader = csv.reader(csvfile)` 修改成 `reader = csv.DictReader(csvfile)`。现在你的代码应该是这样的：

```
import csv

csvfile = open('data-text.csv', 'rb')
reader = csv.DictReader(csvfile)

for row in reader:
    print row
```

保存文件并重新运行，每一个数据记录变成一个字典。字典的键来自于 CSV 文件的第一行。后面所有行都是字典的值。下面是一行数据对应的输出：

```
{
    'Indicator': 'Healthy life expectancy (HALE) at birth (years)',
    'Country': 'Zimbabwe',
    'Comments': '',
    'Display Value': '49',
    'World Bank income group': 'Low-income',
    'Numeric': '49.00000',
    'Sex': 'Female',
    'High': '',
    'Low': '',
    'Year': '2012',
    'WHO region': 'Africa',
    'PUBLISH STATES': 'Published'
}
```

现在我们已经成功地将 CSV 数据导入到 Python 中，也就是说，我们能够从文件中获取数据，并将其转换成 Python 可用的格式（字典），`for` 循环可以让我们直观地查看数据。我们可以利用 `csv` 库两种不同的 reader 来查看数据，一种是列表形式，一种是字典形式。在开始探索和分析数据集时，我们会再次用到这个库。下面继续学习导入 JSON 数据。

3.2　JSON数据

JSON 数据是数据传输最常用的格式之一。人们喜欢这一格式，是因为它结构清晰、易于阅读且方便解析。网站在向页面的 JavaScript 传输数据时，JSON 也是最常用的数据格式之一。许多网站都提供了支持 JSON 的 API，我们会在第 13 章讲到。本节会继续使用全球预期寿命的数据。WHO 并没有提供这一数据的 JSON 格式，但我们为本书创建了 JSON 版本的数据，你可以在代码仓库（https://github.com/jackiekazil/data-wrangling）中找到。

> 如果文件的扩展名是 .json，那里面包含的可能是 JSON 数据。如果文件扩展名是 .js，那可能是 JavaScript 文件，但在少数情况下也可能是命名不规范的 JSON 文件。

在代码编辑器里打开这个 JSON 文件，你会发现每一条数据记录都很像一个 Python 字典。每一行都有键和值，用 : 分隔，数据条目之间用 , 分隔。首尾还有花括号包围 {}。这是 JSON 文件的一条数据记录：

```
[
  {
    "Indicator":"Life expectancy at birth (years)",
    "PUBLISH STATES":"Published",
    "Year":1990,
    "WHO region":"Europe",
    "World Bank income group":"High-income",
    "Country":"Andorra",
    "Sex":"Both sexes",
    "Display Value":77,
    "Numeric":77.00000,
    "Low":"",
    "High":"",
    "Comments":""
  },
]
```

JSON 文件有时看起来和字典完全相同，这与输出格式有关。在上面的示例中，每一个数据条目就是一个 Python 字典（首尾由 { 和 } 包围），这些字典又包含在一个列表中，列表首尾由 [和] 包围。

如何导入JSON数据

在 Python 中导入 JSON 文件比导入 CSV 文件还要简单。下面的代码将对一个 JSON 数据文件执行打开、加载、导入与输出的操作：

```
import json                                  ❶

json_data = open('data-text.json').read()    ❷

data = json.loads(json_data)                 ❸

for item in data:                            ❹
    print item
```

❶ 导入 Python 的 json 库（https://docs.python.org/2/library/json.html），我们用它来处理 JSON 文件。

❷ 利用 Python 内置的 open 函数打开 JSON 文件。文件名叫作 data-text.json（这是 open 函数的第一个参数）。本行代码还调用了已打开文件的 read 方法，用来读取该文件，并将读取的内容保存在变量 json_data 中。

❸ 利用 json.loads() 将 JSON 数据载入 Python，并将输出保存在变量 data 中。

❹ 利用 for 循环遍历所有数据，并打印出每一项，这也是本例代码的输出。

在命令行中运行 python import_json_data.py，输出是一个字典，里面包含 JSON 文件中每一条数据记录。这个输出应该和 CSV 的最终输出基本相同。一定要记得将数据文件复制到脚本所在的文件夹，或者将脚本中的文件路径修改为文件实际所在的位置。

在 CSV 一节的最后，我们学习了如何保存文件并在命令行中运行。在本例中，我们从一个空白文件开始，逐步完成这项任务。

首先来快速看一下总体步骤。

(1) 在代码编辑器中创建一个新文件。
(2) 将文件保存为 import_json_data.py，与你的代码位于同一个文件夹下。
(3) 将数据移动（或保存）到代码所在的文件夹。（一定要重命名数据文件，使其与代码 中的文件名相同。本书用的文件名是 data-text.json。）
(4) 回到代码编辑器，import_json_data.py 文件应该还处于打开状态。

我们来通读代码，并将其与导入 CSV 的代码文件作对比。首先，导入 Python 内置的 `json` 库：

```
import json
```

然后用学过的 `open` 函数打开 data-text.json 文件，并调用已打开文件的 `read` 方法：

```
json_data = open('data-text.json').read()
```

在 CSV 文件的例子中，我们并没有调用 `read`。二者的区别在哪里？在 CSV 的例子中，我们以只读方式打开文件；但在 JSON 的例子中，我们读取文件的内容，并将其保存在变量 `json_data` 中。在 CSV 的例子中，`open` 函数返回的是一个文件对象；但在 JSON 的例子中，我们首先打开文件，然后读取文件，所以得到的是一个 `str`（字符串）。二者的不同是基于下列事实：Python 的 `json` 库和 `csv` 库处理输入数据的方式不同。如果你试着将一个字符串传递给 CSV reader，Python 会报错；如果你把文件对象传递给 JSON 的 `loads` 函数，Python 也会报错。

好消息是，在 Python 中将字符串写入文件非常简单（比如说，你只有字符串，但想使用 CSV reader 来读取），将文件读取成字符串也非常简单。对 Python 来说，一个关闭的文件只是一个文件名字符串，等待被打开并读取。从文件中获取数据、将数据变成字符串并将字符串传递给函数，只需要几行 Python 代码即可完成。

进入保存 JSON 文件的文件夹，你可以在 Python 解释器中输入下面的代码，看一下前面两个例子中输出对象的类型：

```
filename = 'data-text.json'
type(open(filename, 'rb'))  # 与csv的代码类似
type(open(filename).read()) # 与json的代码类似
```

Python `json` 库的 `loads` 函数接收字符串作为参数，不接收文件作为参数。Python `csv` 库的 `reader` 函数接收打开的文件作为参数。在脚本的下一行代码中，我们将使用 `loads` 函数，将 JSON 字符串载入 Python。这一函数的输出被赋值给名为 `data` 的变量：

```
data = json.loads(json_data)
```

想要预览数据，我们对每一项进行遍历并将其打印出来。这段代码并不是必需的，但可以帮我们预览数据，检查数据的格式是否正确：

```
for item in data:
    print item
```

写完上述代码后，保存文件并运行。如你所见，在 Python 中打开 JSON 文件并将其转换成由字典组成的列表是非常容易的。下一节将探索更多自定义文件的处理方法。

3.3 XML数据

XML 格式的数据既便于机器读取，也便于人工读取。但是对于本章的数据集来说，预览并理解 CSV 文件和 JSON 文件要比 XML 文件容易得多。幸运的是，数据本身是相同的，我们也比较熟悉。下载预期寿命数据的 XML 版本（下载地址：http://apps.who.int/gho/athena/data/GHO/WHOSIS_000001,WHOSIS_000002.xml?filter=COUNTRY:*;YEAR:2015），并将 XML 文件与本章其他内容保存在同一文件夹下。

> 如果文件的扩展名是 .xml，那么它是 XML 数据。如果文件扩展名是 .html 或 .xhtml，有时也可以用 XML 解析器来解析。

在处理所有数据时，我们先在代码编辑器中打开文件来预览一下。如果你滚动查看一下文件，会发现我们在 CSV 的例子中已经熟悉的数据。但数据看起来又不大一样，因为它用的是 XML 格式，使用了一种叫作*标签*的东西。

> XML 是一种标记语言，也就是说，它具有包含格式化数据的文档结构。XML 文档本质上只是格式特殊的数据文件。

下面的数据片段是我们要处理的 XML 数据的一个样本。在这个例子中，`<Observation />`、`<Dim />` 和 `<Display />` 都是标签。标签（或节点）以层次化和结构化的方式保存数据：

```
<GHO ...>
    <Data>
        <Observation FactID="4543040" Published="true"
        Dataset="CYCU" EffectiveDate="2014-03-27" EndDate="2900-12-31">
            <Dim Category="COUNTRY" Code="SOM"/>
            <Dim Category="REGION" Code="EMR"/>
            <Dim Category="WORLDBANKINCOMEGROUP" Code="WB_LI"/>
            <Dim Category="GHO" Code="WHOSIS_000002"/>
            <Dim Category="YEAR" Code="2012"/>
            <Dim Category="SEX" Code="FMLE"/>
            <Dim Category="PUBLISHSTATE" Code="PUBLISHED"/>
            <Value Numeric="46.00000">
                <Display>46</Display>
            </Value>
        </Observation>
        <Observation FactID="4209598" Published="true"
        Dataset="CYCU" EffectiveDate="2014-03-25" EndDate="2900-12-31">
            <Dim Category="WORLDBANKINCOMEGROUP" Code="WB_HI"/>
            <Dim Category="YEAR" Code="2000"/>
            <Dim Category="SEX" Code="BTSX"/>
```

```
                <Dim Category="COUNTRY" Code="AND"/>
                <Dim Category="REGION" Code="EUR"/>
                <Dim Category="GHO" Code="WHOSIS_000001"/>
                <Dim Category="PUBLISHSTATE" Code="PUBLISHED"/>
                <Value Numeric="80.00000">
                    <Display>80</Display>
                </Value>
            </Observation>
        </Data>
    </GHO>
```

在 XML 文件中有两个位置可以保存数据值：一个位置是在两个标签之间，比如在 `<Display>46</Display>` 中，`<Display>` 标签的值是 `46`；另一个位置是标签的*属性*，比如在 `<Dim Category="COUNTRY" Code="SOM"/>` 中，`Category` 属性的值是 `"COUNTRY"`，`Code` 属性的值是 `"SOM"`。XML 属性可以保存特定标签的额外信息，这些标签又嵌套在另一个标签中。

在 JSON 中你可以用键 / 值对来保存数据，而在 XML 中保存数据可以是两个一组甚至三四个一组。XML 用标签和属性来保存数据，类似于 JSON 中的键。所以我们再来看一下 `Display` 标签，这个标签的值保存在开始标签和结束标签之间。再来看一下 `Dim` 节点，它有两个不同的属性（`Category` 和 `Code`），两个属性都有对应的值。XML 可以在每个节点中保存不止一个属性。如果你熟悉 HTML 的话，应该很熟悉这一点。这是因为 HTML 与 XML 密切相关：它们都在节点（或标签）内包含有属性，它们也都是标记语言（想了解什么是标记语言，可以去 https://en.wikipedia.org/wiki/Markup_language 查看）。

在 XML 标签结构和属性命名方面虽然有许多著名的标准，但主要结构其实是由设计或创建 XML 的人（或机器）决定的。如果你用的是来自不同来源的数据集，你不能认为它们的格式是一致的。想了解 XML 最佳实践的更多内容，IBM 给出了许多好的观点（参见 http://www.ibm.com/developerworks/library/x-eleatt/）。

如何导入XML数据

前面我们对数据已经有了一定的了解，下面将文件导入成 Python 可用的格式。为了从 XML 格式中提取数据并导入 Python，需要编写这些代码：

```
from xml.etree import ElementTree as ET

tree = ET.parse('data-text.xml')
root = tree.getroot()

data = root.find('Data')

all_data = []

for observation in data:
    record = {}
    for item in observation:
```

```
        lookup_key = item.attrib.keys()[0]

        if lookup_key == 'Numeric':
            rec_key = 'NUMERIC'
            rec_value = item.attrib['Numeric']
        else:
            rec_key = item.attrib[lookup_key]
            rec_value = item.attrib['Code']

        record[rec_key] = rec_value

    all_data.append(record)

print all_data
```

可以看出来，这比 CSV 和 JSON 的代码都要复杂一些。

我们来仔细看一下。在代码编辑器中创建一个新文件，并将其保存在之前保存代码的文件夹中。文件名叫作 import_xml_data.py。另外，如果你是从 WHO 网站直接下载的数据，而不是从本书的仓库中下载的，你需要将保存的 XML 文件重命名为 data-text.xml，并将它与代码放在同一个文件夹下。

首先导入 `ElementTree`（https://docs.python.org/2/library/xml.etree.elementtree.html），这是我们用来解析 XML 的内置库的一部分：

```
from xml.etree import ElementTree as ET
```

> 前面说过，解决问题的方法往往有许多种。本例中用的是 `ElementTree`，你也可以用一个叫作 `lxml` 的库（http://lxml.de/），或者另一个叫作 `minidom` 的库（https://docs.python.org/2/library/xml.dom.minidom.html）。这三种方法都可以用来解决相同的问题，如果你发现一个很好的例子，用的是其中一个库，我们建议你再用另外一个库对数据进行探索。在学习 Python 的过程中，选择那些看起来最容易理解的库（在多数情况下，这也是最好的选择）。

与之前相比，这个 `import` 语句中多了一段：`as ET`。我们导入的是 `ElementTree`，但把它叫作 `ET`。为什么要这么做？因为我们很懒，不想每次用到这个库时都输入 `ElementTree`。在导入名字很长的类或函数时，这是很常见的做法，但并不是强制性的要求。`as` 告诉 Python，我们想用 `ET` 来代表 `ElementTree`。

接下来，调用 `ET` 类的 `parse` 方法，这一方法将会对我们传入文件中的数据进行解析。由于我们要解析的文件位于同一文件夹下，所以文件名中不需要包含文件路径：

```
tree = ET.parse('data-text.xml')
```

`parse` 方法返回一个 Python 对象，人们一般会把它保存在变量 `tree` 中。在谈论 XML 时，树（tree）指的是整个 XML 对象，以 Python 能够理解并解析的方式保存。

为了理解遍历树（及其包含的数据）的方法，我们从树的根元素（root）开始。根节点是第一个 XML 标签。调用 `getroot` 函数来获取树的根元素：

```
root = tree.getroot()
```

如果你在上一条语句后面加上 `print root`，打印出 `root` 的内容，会发现，输出的是 XML 树中根元素的 Python 表示（看起来应该像这样：`<Element 'GHO' at 0x1079e79d0>`[2]）。从这个表示中我们很快可以看出，`ElementTree` 找出了 XML 文档的根标签或最外层标签，就是标签名为 `GHO` 的 XML 节点。

前面我们找到了根标签，下面要学习如何访问想要的数据。在本章 CSV 和 JSON 两节中，我们对数据进行了分析，知道要处理的是什么数据。我们需要遍历整个 XML 树，将同样的数据提取出来。为了理解要寻找的数据，需要先理解 XML 树的整体结构与格式。下面，将前面的 XML 文件简化一下，删除数据，这样就可以只看核心结构：

```
<GHO>
    <Data>
        <Observation>
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Value>
                <Display>
                </Display>
            </Value>
        </Observation>
        <Observation>
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Dim />
            <Value>
                <Display>
                </Display>
            </Value>
        </Observation>
    </Data>
</GHO>
```

在纵览文件结构时可以看到，每一“行”数据都包含在一个 `Observation` 标签中。在每一个 `Observation` 节点内，这些数据行又分别包含在 `Dim`、`Value` 和 `Display` 节点中。

目前我们有三行代码。为了研究如何用 Python 将这些节点提取出来，在现有代码的末尾加上 `print dir(root)`，然后保存文件并在命令行中运行：

```
python import_xml_data.py
```

你会看到变量 `root` 所有的方法和属性。你的代码应该是这样的：

注 2：对于由十六进制数字组成的长字符串表示的 Python 对象，这是 Python 显示内存地址信息的方法。我们的数据处理过程用不到这些内容，所以如果你的内存地址与我们的有所不同，请不必在意。

```python
from xml.etree import ElementTree as ET

tree = ET.parse('data-text.xml')
root = tree.getroot()

print dir(root)
```

运行该文件，你应该会看到下面的输出：

```
['__class__', '__delattr__', '__delitem__', '__dict__', '__doc__',
'__format__', '__getattribute__', '__getitem__', '__hash__', '__init__',
'__len__', '__module__', '__new__', '__nonzero__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', '_children', 'append', 'attrib',
'clear', 'copy', 'extend', 'find', 'findall', 'findtext', 'get',
'getchildren', 'getiterator', 'insert', 'items', 'iter', 'iterfind',
'itertext', 'keys', 'makeelement', 'remove', 'set', 'tag', 'tail', 'text']
```

假设文件太大，无法打开，我们也不知道文件的结构。在处理大型 XML 数据集时经常会遇到这样的问题。我们能怎么做？首先调用 `dir(root)` 来查看 `root` 对象都有哪些方法。我们注意到 `getchildren` 方法，也许可以用来查看 `Observation` 节点的子元素。在查阅官方最新文档（https://docs.python.org/2/library/xml.etree.elementtree.html#xml.etree.ElementTree.Element.getchildren）以及 Stack Overflow 上的一个问题（http://stackoverflow.com/questions/10408927/how-to-get-all-sub-elements-of-an-element-tree-with-python-elementtree）之后，我们发现，`getchildren` 方法可以返回子元素，但官方文档不建议继续使用该方法。如果你想用的某个方法已经不建议使用或者即将弃用，你应该尝试换用库作者推荐的方法。

> 如果不建议使用某个方法、类或函数的话，在库或模块的未来版本中很可能会删除它们对应的功能。因此，在任何时候你都应该避免使用不建议使用的方法或类，并通读文档，因为作者很可能会建议未来使用的替代方法或类。

根据官方文档的建议，如果想查看根元素的子元素，应该使用 `list(root)`。如果我们的文件很大，返回子元素可以让我们查看数据及其结构，而不会产生超级长的输出。让我们来试一下。

将这行代码：

```python
print dir(root)
```

替换成：

```python
print list(root)
```

在命令行中再次运行该文件。你应该会得到下面的输出，是一个由 `Element` 对象构成的列表（对于本例来说，*元素*指的是 XML 节点）：

```
[<Element 'QueryParameter' at 0x101bfd290>,
<Element 'QueryParameter' at 0x101bfd350>,
<Element 'QueryParameter' at 0x101bfd410>,
```

```
<Element 'QueryParameter' at 0x101bfd590>,
<Element 'QueryParameter' at 0x101bfd610>,
<Element 'QueryParameter' at 0x101bfd650>,
<Element 'Copyright' at 0x101bfd690>,
<Element 'Disclaimer' at 0x101bfd710>,
<Element 'Metadata' at 0x101bfd790>,
<Element 'Data' at 0x102540250>]
```

列表包含的 `Element` 对象分别叫作 `QueryParameter`、`Copyright`、`Disclaimer`、`Metadata` 和 `Data`。我们可以遍历这些元素来探索它们的内容，这样才能更好地理解如何提取我们想要的数据。

在 XML 树里面搜索数据时，`Data` 元素可能是一个很好的出发点。现在我们已经找到了 `Data` 元素，可以重点研究这个子元素。获取 `Data` 元素有好几种方法，这里用 `find` 方法。根元素的 `find` 方法可以利用标签名来搜索子元素。然后，我们就可以获取 `Data` 元素的子元素，看看下一步应该做什么。

将这行代码：

```
print list(root)
```

替换成：

```
data = root.find('Data')

print list(data)
```

> 我们还可以使用 `findall` 方法。`find` 和 `findall` 的区别在于，`find` 返回的是匹配的第一个元素，而 `findall` 返回的是匹配的所有元素。我们知道只有一个 `Data` 元素，所以我们用的是 `find` 而不是 `findall`。如果有不止一个元素，要用 `findall` 方法获取所有匹配元素的列表，然后遍历这些元素。

修改完代码后重新运行该文件，你会看到输出一个超级长的列表，列表由 `Observation` 元素组成。这些是我们的数据点。虽然输出里包含很多信息，但你可以看出它是一个列表，因为最后一个字符是 `]`，这是列表结束的符号。

我们来遍历这个列表中的数据。每个 `Observation` 元素代表一行数据，里面应该会包含更多的信息。我们可以分别遍历这些元素，看看都有什么子元素。对于 Python 中的 `Element` 对象，可以遍历其所有的子元素，就像遍历列表一样。因此，我们可以遍历每一个 `Observation` 元素及其每一个子元素。这是我们第一次使用包含循环的循环，从中我们应该可以知道是否还有更多包含数据的子元素。

> 由于 XML 以节点、子节点和属性的方式保存数据，你会经常采用探索每一个节点和子节点（或者元素和子元素）的方法，直到你掌握了数据的结构，以及如何用 Python 来查看这些数据。

将这行代码：

```
print list(root)
```

替换成：

```
for observation in data:
    for item in observation:
        print item
```

然后重新运行该文件。

输出的是许多 Dim 和 Value 对象。我们尝试几种不同的方法，来探索这些元素中可能包含的内容。Python 的 Element 对象有好几种查看数据的方法。每个 Element 节点都有一个属性 text，可以给出节点内包含的文本。

将这行代码：

```
print item
```

替换成：

```
print item.text
```

然后重新运行该文件。

发生了什么？你得到的返回值应该是许多 None。这是因为很多元素的标签之间没有任何文本，所以这些元素的 item.text 都不存在。我们来看一下数据样本里 <Dim /> 的结构。例如：

```
<Dim Category="YEAR" Code="2000"/>
```

在 Python 中，只有在节点中包含文本的情况下，item.text 才是有用的，像这样：

```
<Dim Category="YEAR">2000</Dim>
```

对于第二个例子，item.text 返回的是 2000。

XML 数据可以有许多种结构。我们需要的信息包含在 XML 里，只是不在一眼就能发现的地方。我们来继续探索。

另一个要查看的地方在子元素中。我们来检查一下里面是否包含子元素。将这行代码：

```
print item.text
```

替换成：

```
print list(item)
```

修改后重新运行代码，从输出中可以看出，某些元素（但不是全部）具有子元素。有意思！这些元素其实是 Value 元素。我们来看一下数据样本中这些元素的结构：

```
<Value>
    <Display>
    </Display>
</Value>
```

如果想探索这些子元素，还需要写一个类似之前写过的循环，来遍历每一个 `Observation` 中的元素。

Python 中的 `Element` 对象还有另一个方法可以调用，叫作 `attrib`，它可以返回每一个节点的属性。在查看 XML 结构时我们已经知道，如果节点的标签之间没有值，那么在标签内通常会有属性。

想要查看节点的属性，将这行代码：

```
print list(item)
```

替换成：

```
print item.attrib
```

重新运行代码，可以看到，输出是由包含在属性中的数据组成的许多字典。我们希望将每一行数据保存成一个字典，而不是将每一个元素及其属性保存在不同的字典中。下面是 `attrib` 输出的一条记录：

```
{'Category': 'PUBLISHSTATE', 'Code': 'PUBLISHED'}
{'Category': 'COUNTRY', 'Code': 'ZWE'}
{'Category': 'WORLDBANKINCOMEGROUP', 'Code': 'WB_LI'}
{'Category': 'YEAR', 'Code': '2012'}
{'Category': 'SEX', 'Code': 'BTSX'}
{'Category': 'GHO', 'Code': 'WHOSIS_000002'}
{'Category': 'REGION', 'Code': 'AFR'}
{'Numeric': '49.00000'}
```

在 CSV 的例子中，我们得到了每条数据记录组成的字典，我门试着把上面的输出转换成类似的格式。XML 数据字典的键稍有不同，因为 WHO 在 XML 数据集中提供的数据与 CSV 数据集并不相同。我们将会把数据转换成下面的格式，但键名可以不同。这基本不会影响我们对数据的使用。

提醒一下，CSV reader 的样本数据记录如下：

```
{
   'Indicator': 'Healthy life expectancy (HALE) at birth (years)',
   'Country': 'Zimbabwe',
   'Comments': '',
   'Display Value': '51',
   'World Bank income group': 'Low-income',
   'Numeric': '51.00000',
   'Sex': 'Female',
   'High': '',
   'Low': '',
   'Year': '2012',
   'WHO region': 'Africa',
   'PUBLISH STATES': 'Published'
}
```

对于样本数据记录，我们希望将 XML 数据转换成下面的样子。在解析完 XML 树之后，我们希望数据的格式就是这样：

```
{
   'COUNTRY': 'ZWE',
   'GHO': 'WHOSIS_000002',
   'Numeric': '49.00000',
   'PUBLISHSTATE': 'PUBLISHED',
   'REGION': 'AFR',
   'SEX': 'BTSX',
   'WORLDBANKINCOMEGROUP': 'WB_LI',
   'YEAR': '2012'
}
```

注意 High 和 Low 字段是缺失的。如果 XML 数据集中这两个字段没有缺失的话，我们会把它们添加到新字典的键中。Display 的值也是缺失的。我们决定不用这个值，因为它和 Numeric 的值相同。

现在你的代码应该是像这样的：

```
from xml.etree import ElementTree as ET

tree = ET.parse('data-text.xml')
root = tree.getroot()

data = root.find('Data')

for observation in data:
    for item in observation:
        print item.attrib
```

想要创建数据结构，首先要为每一条数据记录创建一个空字典。我们向空字典中添加键值对，然后将每一条数据记录添加到一个列表中，这样我们最终的列表中就包含了所有的数据记录（类似于前面的 CSV 数据）。

首先创建用来保存数据的空字典和空列表。在最外层 for 循环上面，添加一行代码：all_data = []，然后将 record = {} 作为 for 循环的第一行，像这样：

```
all_data = []

for observation in data:
    record = {}
    for item in observation:
        print item.attrib
```

现在要找出每一行的键和值，并将其添加到数据记录的字典中。每一次调用 attrib，都会得到包含一个或多个键值对的字典，像这样：

```
{'Category': 'YEAR', 'Code': '2012'}
```

看上去 Category 键的值（这里是 YEAR）应该是新字典的键，而 Code 的值（这里是 2012）应该被设成对应的值。回想第 2 章学过的内容，字典的键应该易于检索（比如 YEAR），字典的值应该包含与键对应的值（比如 2012）。认识到这一点，前面一行将变成：

```
'YEAR': '2012'
```

将代码中的 print item.attrib 修改成 print item.attrib.keys()，然后重新运行该文件：

```
for item in observation:
    print item.attrib.keys()
```

输出的是每一个属性字典的键。我们想查看字典的键，这样才能构建新字典的键值对。我们发现只有两种不同的输出：['Category', 'Code'] 和 ['Numeric']。我们对两种情况分别处理。根据前面的研究，对于同时具有 Category 和 Code 的元素，需要用 Category 的值作为键，用 Code 的值作为值。

要做到这一点，在 item.attrib.keys() 的结尾添加 [0]：

```
for item in observation:
    lookup_key = item.attrib.keys()[0]
    print lookup_key
```

这叫索引（indexing）。返回的是列表的第一个元素。

使用列表索引

对于 Python 中的列表或其他可迭代对象，索引指的是取出列表的第 *n* 个对象。Python 的索引从 0 开始，也就是说，第一个元素的编号是 0，第二个元素是 1，以此类推。由于我们的列表中有一个或两个元素，我们只想要第一个元素，所以在代码中加了 [0]。

回头看一下上一章中狗狗的例子：

```
dog_names = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
```

如果想从列表中提取出 Ellie，那你想要的是列表的第三个元素。由于索引编号从 0 开始，你可以用下面的代码提取出这个元素：

```
dog_names[2]
```

在 Python 解释器中试一下上面的代码，然后尝试提取出 Simon。如果用负数作为索引会怎么样？（提示：负数索引是从列表末尾倒着向前数！）

重新运行代码，输出应该是这样的：

```
Category
Category
Category
Category
Category
Category
Category
Numeric
```

现在我们有了键的名字，下面要寻找键对应的值。我们要用 Category 键的值作为新字典的键。在内层 for 循环中创建一个新变量 rec_key，用来保存 item.attrib[lookup_key] 返回的值：

```
for item in observation:
    lookup_key = item.attrib.keys()[0]
```

```
        rec_key = item.attrib[lookup_key]
        print rec_key
```

修改完成后，在命令行中重新运行代码。对于每一条数据记录，我们得到下面的值：

```
PUBLISHSTATE
COUNTRY
WORLDBANKINCOMEGROUP
YEAR
SEX
GHO
REGION
49.00000
```

看起来都很适合做新字典的键，除了最后一个。这是因为最后一个元素是 Numeric 字典，而不是我们要处理的 Category 字典。如果想保留这些数据供后续使用，需要用 if 语句为这些数值元素创建一种特殊情况。

Python 的 if 语句

if 语句最基本的形式，是用来控制代码流的方法。if 语句是在告诉代码：如果满足该条件，那就执行给定的命令。

if 语句的另一种用法是与 else 一起使用。if-else 语句的意思是：如果满足第一个条件，那么执行相应的命令；但如果不满足该条件，那么执行 else 语句中的命令。

除了 if 和 if-else 之外，你还会将 == 作为比较运算符。== 用来给变量赋值，而 == 用来检验两个值是否相等。另外，!= 用来检验两个值是否不相等。这两个运算符返回的都是布尔值：True 或 False。

在 Python 解释器中试试下面的例子：

```
x = 5

if x == 5:
    print 'x is equal to 5.'
```

你看到了什么？ x == 5 返回的是 True，所以打印出了相应的文字。现在试一下这个：

```
x = 3

if x == 5:
    print 'x is equal to 5.'
else:
    print 'x is not equal to 5.'
```

本例中 x 等于 3，不等于 5，所以你应该会看到 else 代码块中 print 语句的输出。在 Python 中，你可以用 if 和 if-else 语句来帮助控制代码流的逻辑。

当 lookup_key 等于 Numeric 时，我们希望使用 Numeric 作为新字典的键，而不是用它对应的值作为新字典的键（就像 Category 键那样）。将代码修改成：

```
for item in observation:

    lookup_key = item.attrib.keys()[0]
    if lookup_key == 'Numeric':
        rec_key = 'NUMERIC'
    else:
        rec_key = item.attrib[lookup_key]

    print rec_key
```

运行修改后的代码，现在所有的键看起来应该都是我们想要的。下面提取出想要保存在新字典中的值，并将它们与这些键相关联。对于 Numeric，问题比较简单，因为我们只想要 Numeric 键对应的值。对代码作如下修改：

```
    if lookup_key == 'Numeric':
        rec_key = 'NUMERIC'
        rec_value = item.attrib['Numeric']
    else:
        rec_key = item.attrib[lookup_key]
        rec_value = None

    print rec_key, rec_value
```

运行修改后的代码，你会发现 Numeric 的 rec_value 已经匹配好了。比如：

```
NUMERIC 49.00000
```


对于其他所有的值，我们将 rec_value 设置成 None。在 Python 中，None 用来表示一个空值。我们来将这些空值填上真实的数值。记得每一条数据记录都有一个 Category 键和一个 Code 键，像这样：{'Category': 'YEAR', 'Code': '2012'}。对于这些元素，我们想将 Code 的值保存为 rec_value。修改这一行代码：rec_value = None，将 if-else 语句改成下面这样：

```
    if lookup_key == 'Numeric':
        rec_key = 'NUMERIC'
        rec_value = item.attrib['Numeric']
    else:
        rec_key = item.attrib[lookup_key]
        rec_value = item.attrib['Code']

    print rec_key, rec_value
```

重新运行代码，你现在应该会看到，rec_key 和 rec_value 都有相应的值。下面来创建字典：

```
    if lookup_key == 'Numeric':
        rec_key = 'NUMERIC'
        rec_value = item.attrib['Numeric']
    else:
        rec_key = item.attrib[lookup_key]
        rec_value = item.attrib['Code']

    record[rec_key] = rec_value ❶
```

❶ 将每一个键值对添加到 record 字典中。

我们还需要将每一条数据记录添加到 all_data 列表中。在 2.3.3 节中讲过，可以用列表的 append 方法向列表中添加元素。在外层 for 循环的结尾，record 中已经包含了每一个子元素的键，这时我们将 record 添加到列表中。最后，在文件结尾添加 print 来查看数据。

将 XML 树转换成字典的全部代码应该是这样的：

```
from xml.etree import ElementTree as ET

tree = ET.parse('data-text.xml')
root = tree.getroot()

data = root.find('Data')

all_data = []

for observation in data:
    record = {}
    for item in observation:

        lookup_key = item.attrib.keys()[0]

        if lookup_key == 'Numeric':
            rec_key = 'NUMERIC'
            rec_value = item.attrib['Numeric']
        else:
            rec_key = item.attrib[lookup_key]
            rec_value = item.attrib['Code']

        record[rec_key] = rec_value

    all_data.append(record)

print all_data
```

运行上面的代码，你会看到一个长列表，列表的元素是每一条数据记录组成的字典，与 CSV 例子中的相同：

```
{'COUNTRY': 'ZWE', 'REGION': 'AFR', 'WORLDBANKINCOMEGROUP': 'WB_LI',
'NUMERIC': '49.00000', 'SEX': 'BTSX', 'YEAR': '2012',
'PUBLISHSTATE': 'PUBLISHED', 'GHO': 'WHOSIS_000002'}
```

可以看出来，从 XML 中提取数据要稍复杂一些。有时 CSV 文件和 JSON 文件也并不像本章的例子那样容易处理，但它们通常比 XML 文件容易处理一些。但是，作为 Python 开发人员，处理 XML 数据让你可以深入探索并成长，让你可以创建空列表和字典，然后向里面填充数据。在研究如何从 XML 树结构里提取数据的过程中，你还锻炼了自己的调试能力。在通往成为更优秀的数据分析员的路上，这些都是宝贵的经验。

3.4　小结

能够用 Python 处理机器可读的数据格式，这是数据处理的必备技能之一。本章讲了 CSV、JSON 和 XML 三种文件类型。表 3-2 给出了在处理 WHO 数据的不同文件格式时所用到的

Python 库。

表3-2：文件类型和文件扩展名

文件类型	文件扩展名	Python库
CSV、TSV	.csv、.tsv	`csv` 库（https://docs.python.org/2/library/csv.html）
JSON	.json、.js	`json` 库（https://docs.python.org/2/library/json.html）

我们还讲了一些新的 Python 概念。现在你应该知道如何在 Python 解释器中运行 Python 代码，以及如何将代码保存到新文件，并在命令行中运行。我们还学习了用 `import` 导入文件，以及用 Python 的 `read` 和 `open` 打开本地文件并读取。

我们讲的编程新概念还包括用 `for` 循环遍历文件、列表或树，还有用 `if-else` 语句判断特定条件是否满足，然后据此执行对应的命令。表 3-3 对本章学过的新函数和代码逻辑做了总结。

表3-3：Python编程的新概念

概念	作用
`import`（https://docs.python.org/2/reference/simple_stmts.html#import）	向 Python 中导入模块
`open`（https://docs.python.org/2/library/functions.html#open）	内置函数，用 Python 打开本地文件
`for` 循环（http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/loops.html#basic-for-loops）	一段代码，运行 *n* 次
`if-else` 语句（http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html#simple-if-statements）	如果满足特定条件，运行一段代码
`==`（等于运算符，https://docs.python.org/2/reference/expressions.html#not-in）	检验两个值是否相等
序列索引（https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange）	取出序列（字符串、列表等）中第 *n* 个对象

最后，我们在本章创建并保存了许多代码文件和数据文件。假如你完成了本章的所有练习，应该有三个代码文件和三个数据文件。本章前面推荐过组织代码的方法。如果你还没有照做的话，现在马上去做。这是目前所有文件的组织结构示例：

```
data_wrangling/
    code/
        ch3_easy_data/
            import_csv_data.py
            import_xml_data.py
            import_json_data.py
            data-text.csv
            data-text.xml
            data-json.json
        ch4_hard_data/
            ...
```

接下来，我们要学习更复杂的数据格式！

第 4 章

处理Excel文件

与上一章的数据不同，将本章和下一章的数据导入 Python 不会那么轻松，某些数据的导入需要花点工夫。这是因为有些数据格式是用于机器读取的，而另一些数据格式是通过桌面工具来交互的，比如我们即将看到的那些数据。在本章和下一章里，我们将研究两种文件类型实例：Excel 文件和 PDF，并给出几条一般性说明，在遇到其他文件类型时可以参考。

目前为止，你在本书中学到的数据导入方法都是比较常规的方法。本章我们将开始学习一些数据处理过程，每次处理过程之间都会有很大差异。虽然过程更加困难，但最终目标是相同的：提取有用信息，并将其转换成 Python 可用的格式。

本章和下一章的例子中使用的数据来自于 UNICEF（联合国儿童基金会）2014 年的报告，报告主题是“世界儿童状况”（http://www.unicef.org/sowc2014/numbers/）。数据有 PDF 和 Excel 两种格式。

当需要从这些更难处理的文件格式中提取数据时，你可以想象文件里面有一个很恨你的人，因为过程可能很痛苦。我们向你保证，在大多数情况下，生成数据文件的人只是不知道发布机器可读格式文件的重要性。

4.1　安装Python包

首先我们要学习如何安装 Python 外部包（或库）。目前为止，我们使用的 Python 库都是在安装 Python 时默认安装的。还记得在第 3 章里我们导入的 `csv` 和 `json` 包吗？它们属于标准库里的程序包，在安装 Python 时默认安装。

Python 默认安装了一些常用库。由于许多库并不常用，所以你需要手动指定安装。如果把 Python 所有的库都装到电脑里，占用的空间会很大。

Python 库有一个汇总在线目录，叫作 PyPI（https://pypi.python.org/pypi），里面保存了大量的 Python 包及其元数据和文档。

本章我们要处理的是 Excel 文件。在浏览器中访问 PyPI 网站，你可以搜索与 Excel 相关的库（搜索结果见 https://pypi.python.org/pypi?:action=search&term=excel&submit=search），搜索结果中有许多可以下载的 Python 包。这是搜索应该使用哪个 Python 包的一种方法。

从现在开始，我们将使用 pip 来安装 Python 包。安装 pip（安装方法见 https://pip.pypa.io/en/latest/installing/#install-pip）的方法有很多种，你在第 1 章里应该已经安装好了。

首先，要找出 Excel 中的数据值。我们通过安装外部包 `xlrd`（https://pypi.python.org/pypi/xlrd/0.9.3）来实现。我们用 pip 安装：

```
pip install xlrd
```

运行 `uninstall` 命令可以卸载这个 Python 包：

```
pip uninstall xlrd
```

试一下安装 `xlrd`，然后卸载，然后重新安装。掌握 `pip` 命令是很有用的，因为在本书中和你的数据处理生涯中都会经常用到这些命令。

有那么多可选的 Python 包，为什么选择 `xlrd` 呢？选择 Python 库的过程是不完善的。挑选的方法有许多种。不要试图去找到正确的库。在磨炼自身技能的过程中，你可能需要从几个库中选择，选择你能理解的那个库。

我们建议，首先要去网络上搜索，看看其他人推荐了哪些库。如果搜索“用 python 解析 excel”（https://www.google.com.sg/search?q=parse+excel+using+python&oq=parse+excel+using+python&gws_rd=cr&ei=Zu7gV-L_FYqEmgH_zKT4Cg），你会在发现搜索结果前几条里就有 `xlrd` 库。

但是答案并不总是这么明显。在第 13 章研究 Twitter 库时，我们将学习更多关于选择过程的内容。

4.2　解析Excel文件

想从 Excel 工作表中提取数据，有时最简单的方式反而是寻找更好的方法来获取数据。直接解析有时并不能解决问题。在开始解析文件之前，先回答下面几个问题。

- 你是否尝试过寻找其他格式的数据？有时同一个数据源可能也会提供其他格式的数据。
- 你是否尝试过电话咨询，询问数据是否还有其他格式？在第 6 章中给出了更多建议。
- 你是否尝试过将 Excel 文件（或文档阅读器）的一个或多个标签导出成 CSV 格式？如果你的 Excel 工作表中只有几个标签里有数据，或者只有一个标签里的独立数据，这是一种很好的解决办法。

如果这些方法你都试过了，还是找不到你想要的数据，那你就需要用 Python 来解析 Excel 文件。

4.3 开始解析

我们解析 Excel 文件用的是 xlrd 库。这个库是用 Python 处理 Excel 文件的一系列库（http://www.python-excel.org/）之一。

处理 Excel 文件主要有三个库。

- xlrd
  读取 Excel 文件。
- xlwt
  向 Excel 文件写入，并设置格式。
- xlutils
  一组 Excel 高级操作工具（需要先安装 xlrd 和 xlwt）。

在用到这三个库的时候你需要分别安装。但本章只会用到 xlrd。我们要将 Excel 文件读取到 Python 中，所以你先要检查是否安装了 xlrd：

```
pip install xlrd
```

如果你得到以下错误信息，说明你没有安装 pip：

```
bash: pip: command not found
```

有关 pip 的安装说明，可以查阅 1.2.4 节，也可以访问 https://pip.pypa.io/en/latest/installing/。

按步骤完成以下内容，即可安装好 Excel 文件的工作环境（也可能是类似的步骤，这和你的文件组织系统有关）。

(1) 为 Excel 任务创建一个文件夹。
(2) 创建一个新的 Python 文件，文件名叫 parse_excel.py，把它放到上面创建的文件夹中。
(3) 从本书仓库（https://github.com/jackiekazil/data-wrangling）下载名为 SOWC 2014 Stat Tables_Table 9.xlsx 的 Excel 文件，放到同一个文件夹中。

进入这个文件夹，在终端中输入下面的命令，从命令行中运行该脚本：

```
python parse_excel.py
```

学完本章后，我们写的脚本就可以解析这个 Excel 文件中保存的童工数据和童婚数据。

在脚本文件的开头，我们需要导入 xlrd 库，然后用 Python 打开 Excel 工作簿。我们将打开的文件保存在变量 book 中：

```
import xlrd

book = xlrd.open_workbook('SOWC 2014 Stat Tables_Table 9.xlsx')
```

与 CSV 不同，Excel 工作簿可以有多个标签（tab）或工作表（sheet）。想要获取数据，我

们要找到包含目标数据的工作表。

如果有几个工作表，你可以猜一下索引号，但如果工作表很多的话就没法猜了。所以你应该知道 book.sheet_by_name(*somename*) 命令，其中 *somename* 是你要访问工作表的名字。

我们来看一下工作表都有哪些名字：

```
import xlrd

book = xlrd.open_workbook('SOWC 2014 Stat Tables_Table 9.xlsx')

for sheet in book.sheets():
    print sheet.name
```

我们要找的工作表是 Table 9。所以我们把这个名字添加到脚本中：

```
import xlrd

book = xlrd.open_workbook('SOWC 2014 Stat Tables_Table 9.xlsx')
sheet = book.sheet_by_name('Table 9')

print sheet
```

运行上面的代码，程序会退出，并给出以下错误信息：

```
xlrd.biffh.XLRDError: No sheet named <'Table 9'>
```

这时你可能有些困惑不解。问题在于我们看到的内容与实际内容并不相同。

打开 Excel 工作簿，双击工作表名选中它，你会发现在结尾有一个多出来的空格。用户在浏览时是看不到这个空格的。第 7 章中我们将学习如何用 Python 处理这样的错误。现在我们暂且在代码中加一个空格。

将这行代码：

```
sheet = book.sheet_by_name('Table 9')
```

修改成：

```
sheet = book.sheet_by_name('Table 9 ')
```

现在运行脚本应该一切正常了。你会看到类似这样的输出：

```
<xlrd.sheet.Sheet object at 0x102a575d0>
```

我们来探索一下能对工作表做哪些事情。在给 sheet 变量赋值后，添加这行代码，然后重新运行脚本：

```
print dir(sheet)
```

在返回的列表中，你会发现一个叫作 nrows 的方法。我们将用这个方法来遍历所有行。如果加上 print sheet.nrows，返回值是总行数。

现在试一下：

```
print sheet.nrows
```

你得到的返回值应该是 303。我们要遍历每一行，也就是说我们需要一个 for 循环。在 3.1.1 节我们学过，for 循环可以遍历列表中的元素，所以我们需要将 303 转换成一个列表，这样就可以遍历 303 次。我们将使用 range 函数来实现。

> **什么是 range()？**
>
> 还记得我们说过，Python 有许多有用的内置函数吗？range 就是其中之一。range 函数（https://docs.python.org/2/library/functions.html#range）接收一个数字作为参数，输出一个那么多元素组成的列表。
>
> 打开 Python 解释器，输入下面的代码，看看 range 函数的输出是什么样的：
>
> ```
> range(3)
> ```
>
> 输出应该是：
>
> ```
> [0, 1, 2]
> ```
>
> 返回了三个元素。现在我们可以创建一个 for 循环对列表遍历三次。
>
> 关于 range 函数需要注意以下两点。
>
> - 返回的列表是从 0 开始的。这是因为 Python 列表索引是从 0 开始的。如果你想让列表从 1 开始，可以设置范围的始末。比如，range(1, 4) 返回的是 [1, 2, 3]。注意，列表中不包含最后一个数字，如果想得到的是 [1, 2, 3]，需要将第二个数字设为 4。
> - Python 2.7 还有一个叫作 xrange 的函数。二者略有不同，但只有在处理大型数据集时才会发现——xrange 的速度要更快一些。

有了 range 函数，就可以将 303 转换成一个列表，用于 for 循环的遍历，我们的脚本应该是像这样的：

```
import xlrd

book = xlrd.open_workbook('SOWC 2014 Stat Tables_Table 9.xlsx')
sheet = book.sheet_by_name('Table 9 ')

for i in range(sheet.nrows):        ❶
    print i                         ❷
```

❶ 对 range(303) 的索引号 i 做循环，range(303) 是一个包含 303 个连续整数的列表。
❷ 输出 i，是从 0 到 302 的连续整数。

下面需要查找每一行，提取出每行的内容，而不是仅仅打印编号。要实现查找功能，需要用 i 作为索引编号来定位第 *n* 行。

我们用 row_values 来获取每一行的值，这个方法是之前 dir(sheet) 给出的。根据 row_values 的文档（http://www.lexicon.net/sjmachin/xlrd.html#xlrd.Sheet.row_values-method），我们知道这个方法接收一个索引数字，返回对应行的值。将这个方法添加到 for 循环中，重新运行脚本：

```
for i in range(sheet.nrows):
    print sheet.row_values(i)       ❶
```

❶ 利用 `i` 作为索引来查找对应行的值。由于这个方法是在 `for` 循环中，`for` 循环的次数等于工作表的长度，所以我们对数据表的每一行都调用了这个方法。

运行代码，每行都输出一个列表。下面给出的是一部分输出：

```
['', u'TABLE 9. CHILD PROTECTION', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '']
['', '', u'TABLEAU 9. PROTECTION DE L\u2019ENFANT', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', u'TABLA 9. PROTECCI\xd3N INFANTIL', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '']
['', u'Countries and areas', '', '', u'Child labour (%)+\n2005\u20132012*', '',
'', '', '', '', u'Child marriage (%)\n2005\u20132012*', '', '', '', u'Birth
registration (%)+\n2005\u20132012*', '', u'Female genital mutilation/ cutting (%)+\
n2002\u20132012*', '', '', '', '', '', u'Justification of wife beating (%)\n 2005\
u20132012*', '', '', '', u'Violent discipline (%)+\n2005\u20132012*', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '']
```

现在我们能够查看每一行的内容，我们需要从中提取出需要的信息。为了确定需要哪些信息，以及如何获取这些信息，更简单的做法是用 Excel 程序打开该文件，比如 Windows 上的 Microsoft Excel 或 Mac 上的 Numbers。打开工作簿的第二个标签，你会看到好多标题行。

> 我们代码的目的是抓取英文文本。但如果你想挑战一下自己，可以尝试提取法语或西班牙语的标题和国家。

在第二个标签里，看一下你能提取哪些信息，并思考一下如何优化组织这些信息。我们这里提供一种可行的方法，但还有很多其他方法，用到了不同的数据结构。

对于本次练习来说，我们要提取的是童工和童婚的统计数据。下面是组织数据的一种方法——后面将以此例作为目标：

```
{
    u'Afghanistan': {
        'child_labor': {
            'female': [9.6, ''], ❶
            'male': [11.0, ''],
            'total': [10.3, '']},
        'child_marriage': {
            'married_by_15': [15.0, ''],
            'married_by_18': [40.4, '']
        }
    },
```

```
        u'Albania': {
            'child_labor': {
                'female': [9.4, u'  '],
                'male': [14.4, u'  '],
                'total': [12.0, u'  ']},
            'child_marriage': {
                'married_by_15': [0.2, ''],
                'married_by_18': [9.6, '']
            }
        },
        ...
    }
```

❶ 如果你查看 Excel 中的数据，有些数字可能不太一样。这是因为 Excel 常常会将数字四舍五入。我们这里给出的数字就是你用 Python 解析时会得到的数字。

> 在开始写代码时，提前想好输出格式的样子，并写出一个这样的数据实例，可以节省很多时间。一旦确定了你想要的数据格式，你可以问问自己："下一步我要怎么做才能实现目标？"当你不知道下一步怎么做时，这种方法特别有用。

我们将使用两种 Python 结构来提取数据。用到的第一个方法是*嵌套* `for` 循环，就是在把一个 `for` 循环放到另一个 `for` 循环里。如果你有 *x* 行数据，每行包含 *y* 个元素，经常会用到这种方法。想要访问每一个元素，你需要一个 `for` 循环遍历每一行，然后另一个 `for` 循环遍历每一个元素。我们在第 3 章的例子中也用过嵌套 `for` 循环。

我们将使用嵌套 `for` 循环来输出每一行的每一个单元格。这将会输出我们之前见过的元素，之前每一行是以列表的形式出现。

```
for i in xrange(sheet.nrows):
    row = sheet.row_values(i)        ❶

    for cell in row:                 ❷
        print cell                   ❸
```

❶ 将每一行内容组成的列表保存到 `row` 变量中。这提高了代码的可读性。
❷ 遍历列表中的每一个元素，也就是当前行的每一个单元格。
❸ 输出单元格的值。

运行包含嵌套 `for` 循环的完整代码，你会发现输出不再那么有用了。这时我们要用第二种方法来探索 Excel 文件——*计数器*。

什么是计数器

计数器是一种控制程序流的方法。有了计数器，你可以在 `for` 循环中添加一条 `if` 语句来控制 `for` 循环，每次迭代后计数增加。如果计数超过了你的设定值，`for` 循环将不再运行 `if` 语句中的代码。在 Python 解释器中试试下面的例子：

```
count = 0                        ❶

for i in range(1000):            ❷
    if count < 10:               ❸
      print i
    count += 1                   ❹

print 'Count: ', count           ❺
```

❶ 将变量 count 初始值设为 0。
❷ 创建一个循环，循环范围是从 0 到 999。
❸ 测试计数是否小于 10，如果是的话，输出 i。
❹ 增加 count 的值，这样计数会随着循环次数增加而增加。
❺ 输出最终计数。

在我们的代码中添加一个计数器，这样就可以逐项查看每一行和每一个单元格，找出我们需要提取的内容。一定要注意计数器在代码中的位置——放在单元格的循环里或者放在行的循环里，结果可能会大不相同。

修改 for 循环的代码如下：

```
count = 0
for i in xrange(sheet.nrows):
    if count < 10:
        row = sheet.row_values(i)
        print i, row              ❶

    count += 1
```

❶ 输出行编号 i 和对应行的内容，这样我们可以看到每一行包含的信息。

现在回头看一下我们期望的最终输出格式，我们真正要搞清楚的是国家名字是从哪行开始的。要记住，国家名字是最终输出字典的第一个键：

```
{
    u'Afghanistan': {...},
    u'Albania': {...},
    ...
}
```

上面脚本中含有计数器，并且控制语句是 `count < 10`，运行后你会发现，输出中并没有包含国家名字出现的那一行。

要跳过几行才能找到感兴趣的数据，所以我们要想办法确定从哪一行开始采集数据。在上面的例子中，我们知道国家名字开始的行编号大于 10。但怎么能知道具体是从哪一行开始的呢？

答案就在下一个代码示例中，但在你看答案之前，试着自己修改计数器，找到国家名字开始的那一行。（有许多种方法都可以找到，所以即使你的答案与下面的代码示例稍有不同，也是可以的。）

找到了正确的行编号，你需要在那一行后面添加一条 if 语句，开始提取该行的内容。我们只处理那一行之后的数据。

如果你找到了正确的行编号，代码应该是像这样的：

```
count = 0

for i in xrange(sheet.nrows):
    if count < 20:                      ❶
        if i >= 14:                     ❷
            row = sheet.row_values(i)
            print i, row
        count += 1
```

❶ 这一行代码遍历前 20 行内容，找到国家名字开始出现的那一行。
❷ if 语句的作用是，从国家名字出现的那一行开始输出每行的内容。

现在你的输出应该是像这样的：

```
14 ['', u'Afghanistan', u'Afghanistan', u'Afganist\xe1n', 10.3, '', 11.0, '', 9.6,
'', 15.0, '', 40.4, '', 37.4, '', u'\u2013', '', u'\u2013', '', u'\u2013', '', u'\
u2013', '', 90.2, '', 74.4, '', 74.8, '', 74.1, '', '', '', '', '', '', '', '', '',
'', '', '', '']
15 ['', u'Albania', u'Albanie', u'Albania', 12.0, u' ', 14.4, u' ', 9.4,
u' ', 0.2, '', 9.6, '', 98.6, '', u'\u2013', '', u'\u2013', '', u'\u2013', '',
36.4, '', 29.8, '', 75.1, '', 78.3, '', 71.4, '', '', '', '', '', '', '', '', '',
'', '', '', '']
16 ['', u'Algeria', u'Alg\xe9rie', u'Argelia', 4.7, u'y', 5.5, u'y', 3.9, u'y',
0.1, '', 1.8, '', 99.3, '', u'\u2013', '', u'\u2013', '', u'\u2013', '', u'\u2013',
'', 67.9, '', 87.7, '', 88.8, '', 86.5, '', '', '', '', '', '', '', '', '', '', '',
'', '']
...more
```

下面要把每一行转换成字典格式。这样一来，在后续章节我们想要对数据进行其他操作时，数据将更有意义。

回头看一下前面我们期望的输出格式示例。我们想要的是一个字典，用国家作为键。我们用索引将国家名字提取出来。

什么是索引

回想第 3 章学过的内容，索引是从一系列对象（比如列表）中提取元素的方法。对于要解析的 Excel 文件来说，我们把 i 传入 sheet.row_values()，row_values 方法以 i 作为索引。我们在 Python 解释器里练习一下索引。

创建一个列表实例：

```
x = ['cat', 'dog', 'fish', 'monkey', 'snake']
```

想要提取出第二个元素，你可以添加一个索引编号来指代这个元素，像这样：

```
>>>x[2]
'fish'
```

如果这不是你预期的结果，要记住 Python 的索引编号是从 0 开始的。所以，如果想提取人们一般所说的第二个元素，我们要用数字 1：

```
>>>x[1]
'dog'
```

你还可以用负数索引：

```
>>>x[-2]
'monkey'
```

正数索引和负数索引的区别是什么？你可以发现，正数索引是从前向后数，而负数索引是从后向前数。

切片（slicing）是与索引相关的另一个有用工具。切片可以从一个列表或可迭代对象中“切”出一部分来。比如：

```
>>>x[1:4]
['dog', 'fish', 'monkey']
```

注意，与 `range` 一样，切片操作从第一个数字开始，但第二个数字的意思是：“直到这个数，但并不包括这个数。”

如果你没有输入第一个数或最后一个数，切片操作会一直到头或一直到尾。下面给了几个例子：

```
x[2:]
['fish', 'monkey', 'snake']

x[-2:]
['monkey', 'snake']

x[:2]
['cat', 'dog']

x[:-2]
['cat', 'dog', 'fish']
```

其他可迭代对象的切片操作与列表相同。

我们在代码中添加一个字典，然后提取出每一行的国家名字，作为字典的键。

将 `for` 循环的代码修改如下：

```
count = 0
data = {}                                ❶

for i in xrange(sheet.nrows):
    if count < 10:
        if i >= 14:
            row = sheet.row_values(i)
            country = row[1]             ❷
            data[country] = {}           ❸
            count += 1

print data                               ❹
```

❶ 创建一个空字典来保存数据。
❷ `row[1]` 提取出遍历每一行的国家名字。
❸ `data[country]` 将国家设为 `data` 字典的键，对应的值设为另一个字典，因为我们接下来要将数据保存在这个字典里。
❹ 输出 `data` 字典，我们可以看到里面的内容。

现在你的输出应该是像这样的：

```
{u'Afghanistan': {}, u'Albania': {}, u'Angola': {}, u'Algeria': {},
u'Andorra': {}, u'Austria': {}, u'Australia': {}, u'Antigua and Barbuda': {},
u'Armenia': {}, u'Argentina': {}}
```

现在我们需要将每一行其他的值与工作簿中的值对应起来，然后保存到字典中。

> 在你把所有的值都提取出来，并与 Excel 工作表中的值对比时，你会犯许多错误。这很正常，也是预料之中的事情。你应该接受这个过程，这说明你正在一步步解决问题。

我们首先创建一个空的数据结构，可以用来保存数据。因为我们已经知道数据行是从第 14 行开始的，所以可以删掉计数器。我们知道 `xrange` 可以输入起点和终点，所以可以从 14 开始计数，直到文件的结尾。修改后的代码如下：

```
data = {}

for i in xrange(14, sheet.nrows):     ❶
    row = sheet.row_values(i)
    country = row[1]

    data[country] = {                 ❷
        'child_labor': {              ❸
            'total': [],              ❹
            'male': [],
            'female': [],
        },
        'child_marriage': {
            'married_by_15': [],
            'married_by_18': [],
        }
    }

print data['Afghanistan']             ❺
```

❶ 我们可以删掉计数器相关的代码，让 `for` 循环从工作表第 14 行开始。这一行代码从 `i` 的值为 14 开始循环，所以我们自动跳过数据集中不需要的那些行。
❷ 这一行代码将字典扩展成很多行，可以填入其他数据点。
❸ 这一行代码创建了 `child_labor` 键，并把它的值设为另一个字典。
❹ 这个字典又包含了几个字典，字典的键是字符串，说明了保存的数据内容。每一个键对应的值都是列表。
❺ 输出与 `Afghanistan` 键对应的值。

Afghanistan 的输出数据是像这样的：

```
{
    'child_labor': {'total': [], 'male': [], 'female': []},
    'child_marriage': {'married_by_18': [], 'married_by_15': []}
}
```

现在我们来填入数据。因为我们用索引访问每一行的每一列，所以可以将工作表的值填入这些列表中。观察工作表，弄清楚哪一列对应的是哪一部分数据，我们可以把 data 字典修改成这样：

```
data[country] = {
    'child_labor': {
        'total': [row[4], row[5]],    ❶
        'male': [row[6], row[7]],
        'female': [row[8], row[9]],
    },
    'child_marriage': {
        'married_by_15': [row[10], row[11]],
        'married_by_18': [row[12], row[13]],
    }
}
```

❶ 每一列包含两个单元格，所以我们的代码把两个值都保存下来。这一行代码中童工总数是第 5 列和第 6 列，而我们知道 Python 从 0 开始索引，所以索引编号是 4 和 5。

重新运行代码，我们会得到类似这样的输出：

```
{
  'child_labor': {'female': [9.6, ''], 'male': [11.0, ''], 'total': [10.3, '']},
  'child_marriage': {'married_by_15': [15.0, ''], 'married_by_18': [40.4, '']}}
}
```

在继续学习后面的内容之前，输出几条数据记录，检查字典中的数据是否正确。有时一个索引错了，剩下的数据就全都错了。

最后，我们可以用 pprint 语句而不是 print 语句来预览数据。对于复杂的数据结构（如字典），用这种方法检查输出要容易很多。将下面两行代码添加到文件结尾，可以查看格式化的数据：

```
import pprint              ❶
pprint.pprint(data)        ❷
```

❶ 导入 pprint 库。import 语句一般出现在文件开头，但为了简单起见，我们把它放在这里。后面你可以把这几行删掉，因为它们不会影响脚本的运行。
❷ 将 data 传入 pprint.pprint() 函数。

滚动查看输出的内容，你会发现大部分内容看起来都很好，但有几条记录看起来出了问题。

查看工作表的内容，你应该会注意到，国家的最后一行是津巴布韦（Zimbabwe）。所以我们要找到国家名字是 'Zimbabwe' 的那一行，然后退出循环。我们在代码中添加 break 来退

出循环，这样就完全退出了 for 循环，继续执行下面的脚本。我们来添加 break 停止循环。在 for 循环的结尾，添加下列代码，然后重新运行：

```
if country == 'Zimbabwe':   ❶
    break                   ❷
```

❶ 如果国家名字是津巴布韦的话……
❷ 退出 for 循环。

添加了 break 之后，你有没有得到 NameError: name 'country' is not defined 的错误信息？如果有的话，检查一下代码缩进。for 循环中的 if 语句要缩进四个空格。

逐步运行代码可以帮你发现问题所在。如果你想知道 for 循环中某个变量的值，比如 country，可以在 for 循环中添加 print 语句，在脚本报错退出之前查看变量的值。这样你可能会发现出错的地方在哪里。

现在脚本的输出与我们的最终目标是一致的。我们要做的最后一件事情，是在脚本中添加一些注释。

注　　释

在代码中添加注释，可以让你（或其他人）在以后能够理解代码的含义。添加注释的方法是在注释前加一个 # 号：

```
# 这是Python注释。Python会忽略该行。
```

添加多行注释的格式如下：

```
"""

    这是多行注释的格式。
    如果你的注释很长或者
    你想插入一段较长的描述，
    你应该使用这种类型的注释。

"""
```

你的脚本现在应该是这样的：

```
"""
    这是用来分析童工和童婚数据的脚本。          ❶
    本脚本中用到的Excel文件可以在以下链接中获取：
        http://www.unicef.org/sowc2014/numbers/
"""

import xlrd
book = xlrd.open_workbook('SOWC 2014 Stat Tables_Table 9.xlsx')

sheet = book.sheet_by_name('Table 9 ')
```

```
data = {}
for i in xrange(14, sheet.nrows):
    # 从第14行开始,因为这是国家数据的起点。         ❷

    row = sheet.row_values(i)

    country = row[1]

    data[country] = {
        'child_labor': {
            'total': [row[4], row[5]],
            'male': [row[6], row[7]],
            'female': [row[8], row[9]],
        },
        'child_marriage': {
            'married_by_15': [row[10], row[11]],
            'married_by_18': [row[12], row[13]],
        }
    }

    if country == 'Zimbabwe':
        break

import pprint
pprint.pprint(data)                                ❸
```

❶ 多行注释，大致说明脚本的用途。

❷ 单行注释，说明我们为什么从第 14 行开始，而不是从前面开始。

❸ 从简单的数据解析过渡到数据分析工作时，我们可以也应该删除这两行。

现在我们的输出应该和上一章的数据差不多。在下一章里，我们进一步将相同的数据从 PDF 文件中解析出来。

4.4 小结

Excel 格式是介于机器可读与人工可读之间的奇怪格式，在一定程度上是机器可读的。Excel 文件本来不是用程序来读取的，但可以被程序解析。

为了处理这种非标准格式，我们需要安装外部库。寻找外部库有两种方法：一种方法是搜索 PyPI 网站（https://pypi.python.org/pypi，Python 包索引）；另一种方法是搜索教程和入门指南，看一下其他人的做法。

找到了需要安装的库，可以使用 `pip install` 命令来安装；想要卸载 Python 库，可以使用 `pip uninstall` 命令。

除了学习如何用 `xlrd` 库解析 Excel 文件之外，我们还学习了几个 Python 编程新概念，表 4-1 对这些新概念做了汇总。

表4-1：Python编程新概念

概念	作用
`range` 和 `xrange`（https://docs.python.org/2/library/functions.html#range）	将一个数字转换成连续数字组成的列表。例如：`range(3)` 输出的是 `[0, 1, 2]`
从 0 开始计数，而不是 1	这是需要注意的计算机构造原理（computer construct），所有编程都是这样的。在使用 `range`、索引或切片时一定注意这一点
索引和切片（http://pythoncentral.io/cutting-and-slicing-strings-in-python/）	用来提取一个字符串或列表的特定子集
计数器	用来控制 `for` 循环的工具
嵌套 `for` 循环	用于遍历数据结构中的数据结构，例如由列表组成的列表、由字典组成的列表或者由字典组成的字典
`pprint`	`pprint` 可以在终端中美化输出数据。在编程处理复杂的数据结构时，这个函数很好用
`break`	利用 `break` 可以提前退出 `for` 循环，它可以结束循环，继续执行后面的脚本
注释	在所有代码中添加注释是很重要的，这样你在以后参考的时候能够理解代码的含义

随着继续深入学习处理 PDF 文件，为了回答你要研究的问题，要学会寻找数据的其他格式，或者使用其他寻找数据的方法，这一点是很重要的。

第 5 章

处理PDF文件，以及用Python解决问题

只发布 PDF 格式的数据是十分错误的，但有时你也没有其他选择。本章你将学习如何解析 PDF，在学习过程中，你还会学习如何解决在代码中遇到的错误。

我们还会讲到如何编写脚本，从一些基本概念（例如导入模块）讲起，逐步过渡到一些更复杂的内容。学完本章，你将学会在代码中思考问题与解决问题的许多方法。

5.1　尽量不要用PDF

本章用到的数据与上一章相同，只不过是 PDF 格式的。一般来说，我们不会去寻找难以解析的数据格式，但我们在本书中之所以这么做，是因为你要处理的数据可能并不总是理想中的格式。你可以在本书的 GitHub 仓库（https://github.com/jackiekazil/data-wrangling）中找到本章所用的 PDF 文件。

在开始解析 PDF 数据之前，你需要考虑以下几件事情。

- 你是否尝试寻找其他格式的数据？如果在网上找不到，试试打电话（见 6.4.1 节）或发邮件求助。
- 你是否尝试过从文档中直接复制粘贴数据？有时你可以很方便地在 PDF 文件里选择并复制数据，然后粘贴到电子表格中。但这种做法不一定每次都能奏效，而且也无法规模化（如果有大量文件或页面，你就没法快速完成了）。

如果你不得不处理 PDF 文件的话，需要学习如何用 Python 解析其中的数据。我们来开始学习。

5.2 解析PDF的编程方法

处理 PDF 要比 Excel 文件更加困难，因为每一个 PDF 文件的格式都不可预知。（如果你有一系列 PDF 文件，那么就可以对文件进行解析了，因为这些文档的格式很可能是一致的。）

PDF 工具处理文档的方法有很多种，其中一种方法是将 PDF 转换成文本。在我们写作本书的同时，Danielle Cervantes 在 NICAR 上做了一个关于 PDF 工具的演讲，NICAR 是一个针对记者的 listserv[1]。这个演讲汇总了下列 PDF 解析工具：

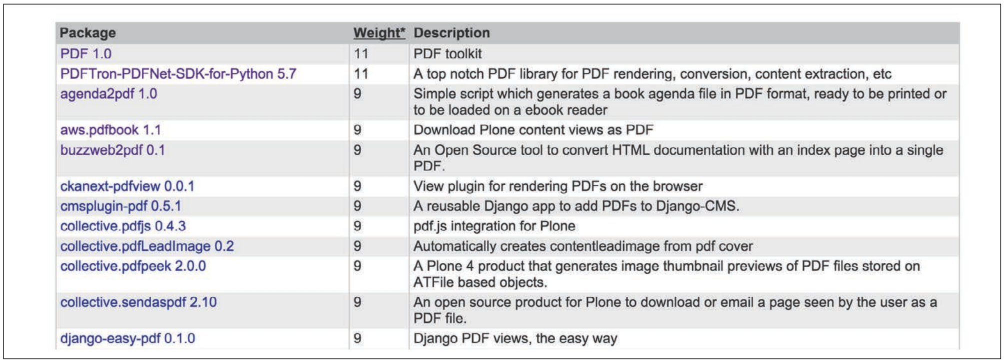
- ABBYY’s Transformer
- Able2ExtractPro
- Acrobat Professional
- Adobe Reader
- Apache Tika
- Cogniview’s PDF to Excel
- CometDocs
- Docsplit
- Nitro Pro
- PDF XChange Viewer
- `pdfminer`
- `pdftk`
- `pdftotext`
- Poppler
- Tabula
- Tesseract
- xPDF
- Zamzar

除了上面这些工具，你还可以用许多编程语言来解析 PDF，其中包括 Python。

> 仅因为你知道类似 Python 这样的工具，并不意味着它总是解决问题的最佳工具。考虑到可用的工具有很多种，你可能会发现用另一个工具来完成部分任务（比如数据提取）更加容易。保持开放的心态，事先对各种可用的工具进行调研。

在 4.1 节中提到过，PyPI 网站是查找 Python 包的好地方。如果搜索“PDF”（https://pypi.python.org/pypi?:action=search&term=pdf&submit=search），你会得到一堆结果，类似图 5-1 给出的那些。

注 1：listserv 是一个用来管理电子邮件列表的软件。——译者注



Package	Weight*	Description
PDF 1.0	11	PDF toolkit
PDFTron-PDFNet-SDK-for-Python 5.7	11	A top notch PDF library for PDF rendering, conversion, content extraction, etc
agenda2pdf 1.0	9	Simple script which generates a book agenda file in PDF format, ready to be printed or to be loaded on a ebook reader
aws.pdfbook 1.1	9	Download Plone content views as PDF
buzzweb2pdf 0.1	9	An Open Source tool to convert HTML documentation with an index page into a single PDF.
ckanext-pdfview 0.0.1	9	View plugin for rendering PDFs on the browser
cmsplugin-pdf 0.5.1	9	A reusable Django app to add PDFs to Django-CMS.
collective.pdfjs 0.4.3	9	pdf.js integration for Plone
collective.pdfLeadImage 0.2	9	Automatically creates contentleadimage from pdf cover
collective.pdfpeek 2.0.0	9	A Plone 4 product that generates image thumbnail previews of PDF files stored on ATFile based objects.
collective.sendaspdf 2.10	9	An open source product for Plone to download or email a page seen by the user as a PDF file.
django-easy-pdf 0.1.0	9	Django PDF views, the easy way

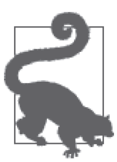
图 5-1：PyPI 网站上的 PDF 包

浏览这些 Python 包，了解一下每个库的详细信息，但分辨不出哪一个库是解析 PDF 的最佳选择。如果你尝试更多的搜索，比如“parse pdf”（解析 pdf），会出现更多的库供你选择，但还是没有明显的最佳选择（搜索结果见 https://pypi.python.org/pypi?:action=search&term=parse+pdf&submit=search）。所以我们用搜索引擎查看一下大家都在用什么库来解析 PDF。



在搜索库或者答案时，注意观察你找到资料的发布日期。帖子或问题的年代越久远，它过时且不再适用的可能性就越大。先试着将搜索范围限定在过去的两年内，然后仅在需要时再扩大搜索的时间范围。

在阅读了许多教程、文档、博客文章和几篇有用的文章（例如这一篇：http://www.binpress.com/tutorial/manipulating-pdfs-with-python/167）之后，我们决定使用 `slate` 库（https://pypi.python.org/pypi/slate）。



`slate` 能够满足我们的需求，但并非总是如此。放弃并从头开始也是可以的。如果有很多库可供选择的话，选择你认为最合适的那一个，即使有人告诉你它不是“最好的”工具。究竟哪一个工具最好，大家见仁见智。在你学习编程的过程中，“最好的”工具就是你凭直觉选择的那一个。

5.2.1 利用slate库打开并读取PDF

我们决定用 `slate` 库来解决这个问题，下面我们来安装这个库。在命令行中运行：

```
pip install slate
```

现在你已经安装了 `slate`，你可以创建一个名为 parse_pdf.py 的脚本，将下面的代码保存其中。一定要确保脚本文件和 PDF 文件位于同一文件夹下，或者修改代码中的文件路径。这段代码会打印出文件的前两行内容：

```
import slate                    ❶

pdf = 'EN-FINAL Table 9.pdf'    ❷
```

```
with open(pdf) as f:            ❸
    doc = slate.PDF(f)          ❹

for page in doc[:2]:            ❺
    print page
```

❶ 导入 slate 库。
❷ 创建字符串变量，用于保存文件路径，一定要确保空格和大小写都正确。
❸ 将文件名字符串传入 Python 的 open 函数，这样 Python 就可以打开该文件。Python 将打开的文件保存为变量 f。
❹ 将打开的文件 f 传递给 slate.PDF(f)，slate 可以将 PDF 文件解析成可用的格式。
❺ 遍历文档 doc 的前两页并输出，这样我们可以知道程序运行正常。

> 通常来说，pip 会安装所有必需的依赖库；但需要包管理器列出这些依赖库。在使用 slate 库或其他库时，如果你得到 ImportError，你应该仔细阅读下一行错误信息，看一下哪个库没有安装。运行代码时如果得到如下错误信息：ImportError: No module named pdfminer.pdfparser，说明安装 slate 时没有正确安装 pdfminer，即使它是必需的库。你需要运行 pip install --upgrade --ignoreinstalled slate==0.3 pdfminer==20110515 来安装 pdfminer（参见 slate 仓库的这个 issue，https://github.com/timClicks/slate/issues/5）。

运行脚本，然后将输出内容与 PDF 中的内容对比一下。

下面是第一页的内容：

```
TABLE 9Afghanistan 10  11  10  15  40  37  -  -  -  -  90  74  75  74Albania
12  14  9  0  10  99  -  -  -  36  30  75  78  71Algeria 5 y 6 y 4 y 0  2  99
-  -  -  -  68  88  89  87Andorra -   -  -  -  -  100 v -  -  -  -  -  -  -
-Angola 24 x 22 x 25 x -  -  36 x -  -  -  -  -  -  -  -Antigua and Barbuda
-   -   -   -  -  -  -  -  -  -  -  -  -  -Argentina 7 y 8 y 5 y -  -  99 y
-  -  -  -  -  -  -  -Armenia 4  5  3  0  7  100  -  -  -  20  9  70  72
67Australia -   -   -   -  -  100 v -  -  -  -  -  -  -  -Austria -   -
-   -  -  100 v -  -  -  -  -  -  -  -Azerbaijan 7 y 8 y 5 y 1  12  94  -  -
-  58  49  75  79  71Bahamas -   -   -   -  -  -  -  -  -  -  -  -  -
-Bahrain 5 x 6 x 3 x -  -  -  -  -  -  -  -  -  -  -Bangladesh 13    18
8    29  65  31  -  -  -  -  33 y -  -  -Barbados -   -   -   -  -  -  -  -
-  -  -  -  -  -Belarus 1    1    2    0  3  100 y -  -  -  4  4  65 y 67 y 62
yBelgium -   -   -   -  -  100 v -  -  -  -  -  -  -  -Belize 6    7    5
3  26  95  -  -  -  -  9  71  71  70Benin 46    47    45    8  34  80  13  2 y
1  14  47  -  -  -Bhutan 3  3  3  6  26  100  -  -  -  -  68  -  -  -Bolivia (
Plurinational    State of) 26 y 28 y 24 y 3  22  76 y -  -  -  -  16  -  -
-Bosnia and Herzegovina 5    7    4    0  4  100  -  -  -  6  5  55  60
50Botswana 9 y 11 y 7 y -  -  72  -  -  -  -  -  -  -  -Brazil 9 y 11 y 6 y
11  36  93 y -  -  -  -  -  -  -  -Brunei Darussalam -   -   -   -  -  -  -
-  -  -  -  -  -  -Bulgaria -   -   -   -  -  100 v -  -  -  -  -  -  -
-Burkina Faso 39  42  36  10  52  77  76  13  9  34  44  83  84  82Burundi
26    26    27    3  20  75  -  -  -  44  73  -  -  -Cabo Verde 3 x,y 4 x,y 3
x,y 3  18  91  -  -  -  16 y 17  -  -  -Cambodia 36 y 36 y 36 y 2  18  62  -
-  -  22 y 46 y -  -  -Cameroon 42    43    40    13  38  61  1  1 y 7  39
47  93  93  93Canada -   -   -   -  -  100 v -  -  -  -  -  -  -  -Central
```

```
African Republic  29    27    30    29  68  61  24  1  11  80 y 80  92  92
92Chad 26    25    28    29  68  16  44  18 y 38  -  62  84  85  84Chile 3 x 3
x 2 x -  -  100 y -  -  -  -  -  -  -  -China -   -   -    -  -  -  -  -  -  -
-  -  -  -  -Colombia 13 y 17 y 9 y 6  23  97  -  -  -  -  -  -  -  -  -Comoros
27 x 26 x 28 x -  -  88 x -  -  -  -  -  -  -  -  -Congo 25    24    25    7  33
91  -  -  -  -  76  -  -  -TABLE 9    CHILD PROTECTIONCountries  and
areasChild labour (%)+ 2005-2012*Child marriage (%) 2005-2012*Birth
registration (%)+ 2005-2012*totalFemale genital mutilation/cutting (%)+
2002-2012*Justification of  wife beating (%) 2005-2012*Violent discipline (%)+
2005-2012*prevalenceattitudestotalmalefemalemarried by 15married by
18womenagirlsbsupport for the practicecmalefemaletotalmalefemale78      THE
STATE OF THE WORLD'S CHILDREN 2014 IN NUMBERS
```

如果你查看一下 PDF 文件，很容易发现页面中每一行的格式。我们来看一下 page 的数据类型是什么：

```
for page in doc[:2]:
    print type(page)          ❶
```

❶ 将代码中的 print page 修改为 print type(page)。

运行代码，输出如下：

```
<type 'str'>
<type 'str'>
```

这样我们知道 slate 中的 page 是一个长字符串。这一点很有用，因为现在我们就可以使用字符串方法（想复习字符串方法，可以回看第 2 章）。

总的来说，读取这个 PDF 文件并不难。由于这个文件中只包含表格，几乎没有任何文本，slate 可以很好地解析。在某些情况下，表格被包围在文本中，你可能需要跳过一些行才能获取你想要的数据。如果你需要跳过一些行的话，可以采用上一章 Excel 例子中的方法，创建一个每行递增的计数器，用来找到数据所在的位置，然后利用 4.3 节“什么是索引”中的方法将我们需要的数据提取出来。

我们的最终目标是从 PDF 文件中提取出数据，数据格式与 Excel 文件的输出格式相同。想要做到这一点，我们需要分割字符串，找出每一行的内容。其背后的思考过程是寻找规律，找到每一个新行开始的标志。这听起来可能很简单，但可能会非常复杂。

在处理大型字符串时，人们通常会使用正则表达式（RegEx）。如果你不熟悉正则表达式和如何编写正则表达式的话，这个方法有点困难。如果你准备挑战一下自己，想学习 Python 中正则表达式的更多内容，可以查看 7.2.6 节。对于我们的目标而言，我们将采用一种更简单的方法来提取数据。

5.2.2　将PDF转换成文本

首先我们希望将 PDF 转换成文本，然后再对文本进行解析。如果文件很大或数量很多，这种方法更好。（使用 slate 库的话，每次运行脚本都要对 PDF 进行解析。如果文件很大或很多的话，这种方法既浪费时间又浪费内存。）

我们要用 `pdfminer` 将 PDF 转换成文本。首先安装这个库：

```
pip install pdfminer
```

安装好 `pdfminer`，你就可以使用一个叫作 `pdf2txt.py` 的命令，将 PDF 文件转换成文本。现在我们来试一下。运行下面的命令，将 PDF 转换成同一文件夹下的文本文件，这样所有的数据都在同一文件夹下：

```
pdf2txt.py -o en-final-table9.txt EN-FINAL\ Table\ 9.pdf
```

第一个参数（`-o en-final-table9.txt`）是我们想要创建的文本文件。第二个参数（`EN-FINAL\ TABLE\ 9.pdf`）是我们的 PDF 文件。确保大小写和文件名中的空格正确。空格前面需要加一个反斜线（\）。这叫转义（escaping）。转义是告诉计算机，空格是输入内容的一部分。

利用 Tab 自动补全

给你介绍一位终端里的新朋友：Tab 键。对于上面命令的第二个参数，你可以先输入 EN，然后按两下 Tab 键。如果只有一种备选的话，计算机会自动补全文件名。如果有多种备选，计算机会发出警告音，并返回一系列备选。在输入又长又奇怪的文件夹或文件名时，这种方法极为有用。

试试这个。切换到 home 目录（在基于 Unix 的系统是 `cd ~/`，在 Windows 上是 `cd %cd%`）。现在，假设你想进入 Documents 目录。试着输入 `cd D`，然后按两下 Tab 键。发生了什么？ home 目录里还有哪些文件或文件夹以字母 D 开头？（可能是 Downloads 文件夹？）

现在试一下 `cd Doc`，然后按两下 Tab 键。你应该会看到自动补全成 Documents 文件夹。

运行完这个命令之后，我们创建了这个 PDF 文件的文本格式文件，叫作 en-final-table9.txt。

我们将新文件读入 Python。创建一个新脚本，与前面的脚本保存在同一文件夹下。脚本名叫作 parse_pdf_text.py，或者你认为合适的其他名字。在脚本中写入下列代码：

```
pdf_txt = 'en-final-table9.txt'
openfile = open(pdf_txt, 'r')

for line in openfile:
    print line
```

我们可以逐行读取文本，然后打印出每一行，将表格内容以文本格式呈现。

5.3 利用pdfminer解析PDF

众所周知，处理 PDF 文件十分困难，我们将学习如何解决在代码中遇到的问题，并掌握一些解决问题的基本方法。

我们希望首先采集国家名称，因为国家名称是最终数据集的键。打开文本文件，你会发现国家出现之前有 8 行。第 8 行的内容是 and areas：

```
5    TABLE 9    CHILD PROTECTION
6
7    Countries
8    and areas
9    Afghanistan
10   Albania
11   Algeria
12   Andorra
```

浏览一下整个文本文档，你会发现相同的规律。因此，我们要创建一个开关变量，在遇到 and areas 这一行时控制采集过程的开始和结束。

为了完成这个任务，我们需要修改 `for` 循环，添加一个布尔变量，即 `True/False` 变量。在遇到 and areas 那一行时将布尔变量设置为 `True`：

```
country_line = False                ❶
for line in openfile:

    if line.startswith('and areas'): ❷
        country_line = True          ❸
```

❶ 将 `country_line` 设置为 `False`，因为默认行里不包含国家。

❷ 搜索以 and area 开头的行。

❸ 将 `country_line` 设置为 `True`。

我们要解决的下一个问题是何时将布尔变量再次设置为 `False`。花点时间查看文本文件，试着找到其中的规律。你怎么能知道国家列表的结尾在哪里呢？

观察下面的文本片段，你会注意到其中有一个空行：

```
45    China
46    Colombia
47    Comoros
48    Congo
49
50    total
51    10
52    12
```

但 Python 怎么识别空行呢？在脚本中添加一行代码，打印出每一行的 Python 表示（查阅 7.2.2 节，可以了解关于字符串格式化的更多内容）：

```
country_line = False
for line in openfile:
    if country_line:                   ❶
        print '%r' % line              ❷

    if line.startswith('and areas'):
        country_line = True
```

❶ 在经历了 `for` 循环的前一次迭代后，如果 `country_line` 的值为 `True`，那么……

❷ ……打印出该行的 Python 表示。

观察输出，你会注意到现在所有行的结尾都多出一些字符：

```
45    'China \n'
46    'Colombia \n'
47    'Comoros \n'
48    'Congo \n'
49    '\n'
50    'total\n'
51    '10   \n'
52    '12   \n'
```

\n 是一行结束的标志，或者叫换行符。我们现在用它作为修改 country_line 变量的标志（marker）。如果 country_line 的值为 True，而 line 的值为 \n，我们就应该将 country_line 设置为 False，因为这一行标志着国名的结束：

```
country_line = False
for line in openfile:

    if country_line:                    ❶
        print line

    if line.startswith('and areas'):
        country_line = True
    elif country_line:
        if line == '\n':                ❷
            country_line = False
```

❶ 如果 country_line 的值为 True，打印出该行，我们可以查看国家名称。这段代码在前，是因为我们不希望它出现在 and areas 测试的后面。我们只想打印出实际的国名，而不想打印出 and areas 这一行。

❷ 如果 country_line 的值为 True，且 line 的值为换行符，则将 country_line 设置为 False，因为国家列表已经结束了。

现在运行代码，返回的似乎是包含国家的所有行。我们最后会将其转换成国家列表。现在，对于我们想要采集的数据，我们要寻找相应的标志，然后重复上面的做法。我们想要的是童工数据和童婚数据。首先来看童工数据，我们需要总数、男童数和女童数。我们先来看总数。

我们将利用相同的方法找出童工总数。

(1) 创建一个 True/False 的开关变量。
(2) 寻找开始标志，将开关变量设置为真。
(3) 寻找结束标志，将开关变量设置为假。

查看一下文本，你会发现数据的开始标志是 total。看一下你所创建文本文件的第 50 行，这里出现了第一个标志 [2]：

注 2：你的文本编辑器很可能可以选择显示行编号，甚至可能可以直接“跳转”到某一行。可以用谷歌搜索一下这些功能的使用方法。

```
45    China
46    Colombia
47    Comoros
48    Congo
49
50    total
51    10
52    12
```

结束标志还是换行符或 `\n`，你可以在 71 行看到：

```
68    6
69    46
70    3
71
72    26   y
73    5
```

我们将这个逻辑添加到代码中，然后用 `print` 查看结果：

```
country_line = total_line = False                ❶
for line in openfile:

    if country_line or total_line:               ❷
        print line

    if line.startswith('and areas'):
        country_line = True
    elif country_line:
        if line == '\n':
            country_line = False

    if line.startswith('total'):                 ❸
        total_line = True
    elif total_line:
        if line == '\n':
            total_line = False
```

❶ 将 `total_line` 设置为 `False`。

❷ 如果 `country_line` 或 `total_line` 的值为 `True`，输出该行的内容，方便我们查看数据。

❸ 找到 `total_line` 的起始点，将 `total_line` 设置为 `True`。本行下面的代码与前面 `country_line` 的代码逻辑相同。

现在我们的代码有一些冗余。我们在重复一些相同的代码，只是开关变量有所不同。这就引出了如何创建非冗余代码的话题。在 Python 中，我们可以用函数来执行重复操作。也就是说，我们可以将这些操作放到一个函数里，然后调用函数执行操作，而不必每次输入全部代码手动执行这些操作。如果我们想测试 PDF 的每一行，可以使用函数。

> 第一次编写函数时，往往不清楚应该将函数放在代码的什么位置。你需要先写好函数的代码，然后再调用函数。这样 Python 就知道函数的功能是什么。

我们将函数命名为 turn_on_off，设置它最多接收 4 个参数。

- line 是我们目前所在的行。
- statue 是一个布尔变量（True 或 False），代表函数的开或关。
- start 是我们要寻找的开始标志——它会触发开或 True 状态。
- end 是我们要寻找的结束标志——它会触发关或 False 状态。

修改代码，将函数框架添加到 for 循环之前。不要忘记添加函数功能的说明——当日后查看这个函数时，你不会一头雾水。这些说明文字叫作文档字符串（docstring）：

```
def turn_on_off(line, status, start, end='\n'):                    ❶
    """
        这个函数用于检查该行是否以特定值开始/结束。                  ❷
        如果该行确实以特定值开始/结束,则状态设为开/关(真/假)。

    """
    return status                                                   ❸

country_line = total_line = False
for line in openfile:
    .....
```

❶ 本行代码是函数的开始，函数最多接收 4 个参数。前三个参数 line、status 和 start 是必需（required）参数，也就是说，由于它们没有默认值，一定要对它们赋值。最后一个参数 end，默认值是换行符，因为这是我们文件的规律。在调用函数时，我们可以传入其他值来替换默认值。

❷ 一定要写函数说明（或文档字符串），这样你才能清楚它的功能。函数说明不必追求完美，只要有就行。以后你可以随时更新。

❸ return 语句是退出函数的正确方法。这个函数返回的是 status，值为 True 或 False。

有默认值的参数要放在最后

在编写函数时，没有默认值的参数一定要放在有默认值的参数前面。这也是上面的例子中 end='\n' 是最后一个参数的原因。我们可以看到，有默认值的参数像是一个键值对（即 *value_name=value*），= 后面即为默认值（在上面的例子中默认值为 \n）。

在函数被调用时，Python 会计算参数的值。如果我们想调用前面关于国家的函数，调用方法是这样的：

```
turn_on_off(line, country_line, 'and areas')
```

这里利用了 end 的默认值。如果你想将默认值替换为两个换行符，可以这样调用：

```
turn_on_off(line, country_line, 'and areas', end='\n\n')
```

假设我们将 status 的默认值设置为 False。我们要如何修改代码？

这是修改前函数的第一行：

```
def turn_on_off(line, status, start, end='\n'):
```

下面给出两种修改方法：

```
def turn_on_off(line, start, end='\n', status=False):
def turn_on_off(line, start, status=False, end='\n'):
```

status 参数要放在必需参数之后。在调用新函数时，我们可以使用 end 和 status 的默认值，也可以用其他值替换：

```
turn_on_off(line, 'and areas')
turn_on_off(line, 'and areas', end='\n\n', status=country_line)
```

如果你不小心把有默认值的参数放在必需参数之前，Python 会报错：SyntaxError: non-default argument follows default argument。你不必记住这句话，但要注意的是，如果你遇到这个错误，要知道它指的是什么意思。

现在把代码从 for 循环中移到函数中。我们想在新的 turn_on_off 函数中复制前面 country_line 的逻辑：

```
def turn_on_off(line, status, start, end='\n'):
    """
        这个函数用于检查该行是否以特定值开始/结束。
        如果该行确实以特定值开始/结束,则状态设为开/关(真/假)。
    """

    if line.startswith(start):                 ❶
        status = True
    elif status:
        if line == end:                        ❷
            status = False
    return status                              ❸
```

❶ 将寻找开始行的标志替换为 start 变量。
❷ 将我们用的结束文字替换为 end 变量。
❸ 基于相同的逻辑，返回 status 变量（end 表示 False，start 表示 True）。

现在我们在 for 循环中调用这个函数，将前面所有代码放在一起之后，脚本是这个样子的：

```
pdf_txt = 'en-final-table9.txt'
openfile = open(pdf_txt, "r")


def turn_on_off(line, status, start, end='\n'):
    """
        这个函数用于检查该行是否以特定值开始/结束。
        如果该行确实以特定值开始/结束,则状态设为开/关(真/假)。
    """
    if line.startswith(start):
        status = True
    elif status:
        if line == end:
            status = False
    return status
```

```
country_line = total_line = False                                   ❶

for line in openfile:
    if country_line or total_line:                                  ❷
        print '%r' % line

    country_line = turn_on_off(line, country_line, 'and areas')     ❸
    total_line = turn_on_off(line, total_line, 'total')             ❹
```

❶ 根据 Python 的语法，一连串 = 符号的意思是，我们将最后一个值赋值给前面每一个变量。本行代码将 False 同时赋值给 country_line 和 total_line。
❷ 我们仍然想要记录在开状态下每一行包含的数据。因此我们使用了 or。Python 中 or 的意思是，如果二者之一为真，执行下面的命令。本行代码的意思是，如果 country_line 和 total_line 有一个值为 True，打印出该行的内容。
❸ 对国家调用函数。将函数返回的状态保存到 country_line 变量中，用于下一次 for 循环。
❹ 对总数调用函数。这一行代码与上一行对国名的用法是相同的。

下面将国家和总数保存成列表。然后将这些列表转换成一个字典，字典的键是国名，字典的值是童工总数。这样我们就可以判断是否需要清洗数据。

创建两个列表的代码如下：

```
countries = []                                                  ❶
totals = []                                                     ❷
country_line = total_line = False
for line in openfile:

    if country_line:                                            ❸
        countries.append(line)                                  ❹
    elif total_line:
        totals.append(line)                                     ❺

    country_line = turn_on_off(line, country_line, 'and areas')
    total_line = turn_on_off(line, total_line, 'total')
```

❶ 创建空的国家列表。
❷ 创建空的总数列表。
❸ 注意我们删除了 if country_line or total_line 语句。下面我们将这条语句分开来写。
❹ 如果该行包含国家，将国家添加到国家列表中。
❺ 这一行采集的是总数，与上一行的采集国家的用法相同。

我们将采用“拉链方法”（zipping）将国家和总数两个数据集合并。zip 函数从每一个列表中取出一个元素，然后将其配对，直到所有的元素全部配对完成。我们可以将合并后的列表传递给 dict 函数，从而将其转换成字典。

在脚本最后添加以下代码：

```
import pprint                                       ❶
test_data = dict(zip(countries, totals))            ❷
```

```
pprint.pprint(test_data)                                         ❸
```

❶ 导入 pprint 库。对于复杂的数据结构，这个库的打印格式可读性更好。

❷ 将国家和总数合并到一起，然后转换成一个字典，将字典保存到一个叫作 test_data 的变量中。

❸ 将 test_data 传递给 pprint.pprint() 函数，以美观的格式打印出数据。

现在运行脚本，你会得到类似这样的一个字典：

```
{'\n': '49   \n',
 ' \n': '\xe2\x80\x93   \n',
 '   Republic of Korea \n': '70   \n',
 '   Republic of) \n': '\xe2\x80\x93   \n',
 '   State of) \n': '37  \n',
 '   of the Congo \n': '\xe2\x80\x93   \n',
 '   the Grenadines \n': '60  \n',
 'Afghanistan \n': '10   \n',
 'Albania \n': '12   \n',
 'Algeria \n': '5  y \n',
 'Andorra \n': '\xe2\x80\x93    \n',
 'Angola \n': '24  x \n',
 'Antigua and Barbuda \n': '\xe2\x80\x93    \n',
 'Argentina \n': '7  y \n',
 'Armenia \n': '4   \n',
 'Australia \n': '\xe2\x80\x93    \n',
 ......
```

现在我们需要做一些数据清洗工作。更详细的内容将会在第 7 章中介绍。现在我们需要做的是清洗字符串，因为它们的可读性很差。我们将创建一个函数来清洗每一行的内容。将这个函数放在 for 循环前面，与另一个函数放在一起：

```
def clean(line):
    """
       清洗代码中的换行符、空格以及其他特殊符号。

    """
    line = line.strip('\n').strip()                             ❶
    line = line.replace('\xe2\x80\x93', '-')                     ❷
    line = line.replace('\xe2\x80\x99', '\'')

    return line                                                  ❸
```

❶ 删除该行中的 \n，然后重新赋值给 line，现在 line 中保存的是清洗后的数据。

❷ 替换特殊字符编码。

❸ 返回清洗后的新字符串。

在上面的数据清洗中，我们可以把方法调用合并在一起，像这样：

```
line = line.strip('\n').strip().replace(
    '\xe2\x80\x93', '-').replace('\xe2\x80\x99s', '\'')
```

然而，想要格式美观，每一行 Python 代码的长度不应超过 80 个字符。这只是一个建议，并不是规定，但控制每行代码的长度可以提高代码的可读性。

下面我们将 clean_line 函数应用到 for 循环中：

```
for line in openfile:
    if country_line:
        countries.append(clean(line))
    elif total_line:
        totals.append(clean(line))
```

现在运行脚本，我们得到的输出更加接近我们的目标：

```
{'Afghanistan': '10',
 'Albania': '12',
 'Algeria': '5  y',
 'Andorra': '-',
 'Angola': '24  x',
 'Antigua and Barbuda': '-',
 'Argentina': '7  y',
 'Armenia': '4',
 'Australia': '-',
 'Austria': '-',
 'Azerbaijan': '7  y',
...
```

浏览一下输出，你会发现我们的方法没能充分解析所有的数据。我们需要找出问题的原因。

名字超过一行的国家似乎被分成了两条数据记录。从玻利维亚（Bolivia）的数据可以发现这一点：我们有两条记录，一条是 'Bolivia (Plurinational': '',，另一条是 'State of)': '26 y',。

在 PDF 文件里可以查看数据的组织结构。你可以在 PDF 中看到图 5-2 中的这几行。

Bhutan	3	3	3
Bolivia (Plurinational State of)	26 y	28 y	24 y
Bosnia and Herzegovina	5	7	4
Botswana	9 y	11 y	7 y
Brazil	9 y	11 y	6 y

图 5-2：PDF 文件中的玻利维亚数据

PDF 文件仿佛兔子洞一般。处理每一个 PDF 文件都要用到特殊的技巧。由于我们对这个 PDF 只需要解析一次，所以我们做了许多人工检查的工作。如果需要定期解析这个 PDF，我们需要仔细查看数据随时间变化的规律，然后用程序来处理这些规律，还要对代码进行检查和测试，确保导入的数据正确。

解决这个问题有两种方法。我们可以创建一个占位符，找到总数里面的空白行，然后将空白行与后面的数据行合并。另一种方法是找出那些国名长度不止一行的国家。由于我们的数据集不是很大，所以我们将尝试第二种方法。

我们将创建一个列表，里面包含每一个跨行国家的第一行内容，然后在脚本中用这个列表来检查每一行。你需要将这个列表放在 for 循环之前。参考元素通常会放在脚本的开头，在必要时方便找到并修改。

将 Bolivia (Plurinational 添加到双行国家组成的列表中：

```
double_lined_countries = [
    'Bolivia (Plurinational',
]
```

现在我们需要修改 for 循环，检查上一行内容是否包含在 double_lined_countries 列表中，如果是的话，将上一行与这一行合并。为此我们需要创建一个 previous_line 变量。然后在 for 循环的结尾对 previous_line 变量赋值。只有这样，在代码进行到循环的下一次迭代时，我们才能将两行合并：

```
countries = []
totals = []
country_line = total_line = False
previous_line = ''                                              ❶

for line in openfile:
    if country_line:
        countries.append(clean(line))
    elif total_line:
        totals.append(clean(line))

    country_line = turn_on_off(line, country_line, 'and areas')
    total_line = turn_on_off(line, total_line, 'total')

    previous_line = line                                        ❷
```

❶ 创建 previous_line 变量，值为空字符串。
❷ 在 for 循环的结尾，将 previous_line 的值修改为当前行的内容。

现在有了 previous_line 变量，我们可以检查 previous_line 是否在 double_lined_countries 列表中，这样我们就知道何时将当前行与上一行合并。另外，我们还要将新合并的一行添加到国家列表中。如果国名的前半部分在 double_lined_countries 列表中的话，一定不要将第一行添加到国名列表中。

根据上面的描述，对代码做如下修改：

```
if country_line:                                              ❶
    if previous_line in double_lined_countries:
        line = ' '.join([clean(previous_line), clean(line)])  ❷
        countries.append(line)
    elif line not in double_lined_countries:                  ❸
        countries.append(clean(line))
```

❶ 我们需要 if country_line 的逻辑，因为它只与国名相关。
❷ 如果 previous_line 在 double_lined_countries 列表中，那么将 previous_line 与当前行合并，并将合并后的内容赋值给 line 变量。你可以看到，join 的作用是利用最前面的字符串将一个字符串列表合并在一起。本行代码使用空格作为连接字符。

❸ 如果该行不在 `double_lined_countries` 列表中，那么将该行内容添加到国家列表中。这里我们用的是 `elif`，在 Python 中的意思是 `else if`。如果你想使用一种不同于 `if - else` 的逻辑流，这是一个很好用的工具。

重新运行脚本，我们发现 `'Bolivia (Plurinational State of)'` 已经合在一起了。现在我们需要检查结果中是否包含了所有国家。由于数据集比较小，我们可以手动检查，但如果数据集较大的话，你需要将检查过程自动化。

数据检查自动化

何时手动检查数据，何时用 Python 自动化完成，怎么选择？下面给出几点建议。

- 如果要定期反复解析数据，选择自动化。
- 如果你的数据集比较大，你很可能应该选择自动化。
- 如果你的数据集可控，而且你只需要解析一次，那么你选哪种方式都可以。在我们的例子中，数据集很小，所以我们没有选择自动化。

用 PDF 阅读器查看 PDF 文件，找出所有占两行的国家名称：

```
Bolivia (Plurinational State of)
Democratic People's Republic of Korea
Democratic Republic of the Congo
Lao People's Democratic Republic
Micronesia (Federated States of)
Saint Vincent and the Grenadines
The former Yugoslav Republic of Macedonia
United Republic of Tanzania
Venezuela (Bolivarian Republic of)
```

我们知道，这些国名的 Python 表示可能不是这样的，所以我们需要将国家打印出来，看看它们的 Python 表示是什么样子，然后将其添加到列表中：

```
if country_line:
    print '%r' % line                                    ❶
    if previous_line in double_lined_countries:
```

❶ 添加 `print '%r'` 语句，输出国名的 Python 表示。

运行脚本，将双行国名的 Python 表示添加到 `double_lined_countries` 列表中：

```
double_lined_countries = [
    'Bolivia (Plurinational \n',
    'Democratic People\xe2\x80\x99s \n',
    'Democratic Republic \n',
    'Micronesia (Federated \n',
    #... 糟糕！
]
```

我们的输出中漏掉了 `Lao People’s Democratic Republic`（老挝人民民主共和国），但它在 PDF 中占了两行。我们打开 PDF 的文本文件，看看问题出在哪里。

看完文本文件，你能发现问题所在吗？再看一下 `turn_on_off` 函数。这个函数的原理与文本的书写方式有什么关系？

问题在于，and areas 之后紧跟着一个空行（`\n`），这正是我们要寻找的标志。查看我们创建的文本文件，你会发现在 1345 行出现了意料之外的空行：

```
...
1343 Countries
1344 and areas
1345
1346 Iceland
1347 India
1348 Indonesia
1349 Iran (Islamic Republic of)
...
```

这说明我们的函数没有正常运行。解决这个问题有好几种方法。对于这个例子来说，我们可以加入更多的代码逻辑，保证开 / 关代码的运行符合预期。开始采集国名时，在结束采集之前应该采集到了至少一个国家。如果一个国家都没有采集到的话，那么我们不应该结束采集过程。我们还可以使用上一行来解决这个问题。我们可以在开 / 关函数中检查上一行代码，确保它不属于某些特殊行。

我们采用增加特殊行的方法，以防遇到其他异常：

```
def turn_on_off(line, status, start, prev_line, end='\n'):
    """
        该函数用于检查该行会是否开始/结束于特定值。
        如果是,且上一行不是特殊行,
        则状态设为开/关(真/假)。
    """
    if line.startswith(start):
        status = True
    elif status:
        if line == end and prev_line != 'and areas':    ❶
            status = False
    return status
```

❶ 如果当前行的值等于 `end`，而且上一行的值不等于 and areas，那么我们可以结束数据采集。这里我们用的是 `!=`，这是 Python 用来测试“不相等”的方法。与 `==` 类似，`!=` 返回的也是布尔值。

你还需要修改调用函数的代码，将 `previous_line` 传入函数：

```
country_line = turn_on_off(line, country_line, 'and areas', previous_line)
total_line = turn_on_off(line, total_line, 'total', previous_line)
```

回到我们最开始的任务——创建双行国家的列表，确保采集到所有双行国家。我们前面进行到了这一步：

```
double_lined_countries = [
    'Bolivia (Plurinational \n',
    'Democratic People\xe2\x80\x99s \n',
```

```
        'Democratic Republic \n',
    ]
```

查看 PDF 文件，我们看到下一个双行国家是 Lao People's Democratic Republic（老挝人民民主共和国）。我们继续将脚本输出的其他双行国家添加到这个列表中：

```
    double_lined_countries = [
        'Bolivia (Plurinational \n',
        'Democratic People\xe2\x80\x99s \n',
        'Democratic Republic \n',
        'Lao People\xe2\x80\x99s Democratic \n',
        'Micronesia (Federated \n',
        'Saint Vincent and \n',
        'The former Yugoslav \n',
        'United Republic \n',
        'Venezuela (Bolivarian \n',
    ]
```

如果你的列表看起来和上面的列表相似的话，运行脚本，你的输出应该找出了所有占两行的国名。一定要在脚本结尾添加 print 语句来查看国家列表：

```
import pprint
pprint.pprint(countries)
```

前面我们在国家列表上花了不少时间，你能想出解决这个问题的其他方法吗？看一下几个双行国家的第二行：

```
'   Republic of Korea \n'
'   Republic \n'
'   of the Congo \n'
```

它们有什么共同点？都以空格开头。用代码检查每一行开头是否有三个空格，这是一种更有效的做法。但是采用前面第一种方法，可以在采集数据的过程中发现数据集的部分缺失。随着你编程水平的不断提高，你将学会解决同一问题的各种方法，然后找出最佳方法。

下面我们看一下童工总数和国家的对应情况。修改 pprint 语句，如下所示：

```
import pprint
data = dict(zip(countries, totals))                     ❶
pprint.pprint(data)                                     ❷
```

❶ 调用 zip(countries, totals)，将国家列表和总数列表合并。这样把两个列表变成了元组。然后我们把元组传递给 dict 函数，将其转换成字典。

❷ 打印出我们刚创建的 data 变量。

返回的是一个字典，国家名称是字典的键，童工总数是字典的值。这并不是我们最终的数据格式。我们这样做是为了查看当前的数据。结果应该是像这样的：

```
 {'': '-',
 'Afghanistan': '10',
  'Albania': '12',
  'Algeria': '5  y',
  'Andorra': '-',
  'Angola': '24  x',
```

```
    ...
}
```

对比 PDF 再次检查这些数据，你会发现，就在双行国家第一次出现的地方，数据出现了错误。对应的数字来自出生登记（Birth registration）一列：

```
{
 ...
 'Bolivia (Plurinational State of)': '',
 'Bosnia and Herzegovina': '37',
 'Botswana': '99',
 'Brazil': '99',
 ...
}
```

如果查看 PDF 的文本文件，你会注意到在双行国家对应的数字那里有一个空行：

```
6
46
3

26  y
5
9  y
```

和采集国名遇到的问题一样，我们要用相同的方法来处理这个数据采集问题。如果我们的数据中有空白行，一定不要把空行采集到数据中，这样我们只采集与国名匹配的数据。修改后的代码如下：

```
for line in openfile:
    if country_line:
        print '%r' % line
        if previous_line in double_lined_countries:
            line = ' '.join([clean(previous_line), clean(line)])
            countries.append(line)
        elif line not in double_lined_countries:
            countries.append(clean(line))

    elif total_line:
        if len(line.replace('\n', '').strip()) > 0:                        ❶
            totals.append(clean(line))

    country_line = turn_on_off(line, country_line,
                               'and areas', previous_line)
    total_line = turn_on_off(line, total_line,
                             'total', previous_line)
    previous_line = line
```

❶ 从经验中我们知道，PDF 文件使用换行符作为空行。本行代码用空字符串替代换行符，并删除空格来清洗数据。然后测试字符串的长度是否仍然大于 0。如果是的话，我们认为里面包含数据，并将其添加到童工总数列表中。

运行修改后的代码，在第一个双行国家那里数据又出现了问题。这次第一个双行国家对应的还是出生登记数据。之后的数值也都是错误的。回来看一下文本文件，找到问题出在哪

里。如果你查看 PDF 文件里对应的那列数据，会发现 PDF 文本文件中的规律是从 1251 行开始的：

```
1250
1251    total
1252    -
1253    5  x
1254    26
1255    -
1266    -
```

进一步观察发现，出生登记列标题的结尾是 total：

```
266     Birth
267     registration
268     (%)+
269     2005-2012*
270     total
271     37
272     99
```

目前搜集童工总数的函数找寻的是 total 这个词，所以在找到下一行国家之前，我们先找到了这一列数据。我们还发现*暴力惩戒比例*［Violent discipline（%）］列也有一个 total 标签，上面有一个空行。这和我们要采集的 total 具有相同的规律。

接二连三地遇到 bug，说明你的代码逻辑可能存在问题。我们的脚本最开始用的是*开 / 关*函数，所以想要从根本解决问题，就要重构那里的逻辑。我们想要知道如何找到正确的数据列，或许可以采集列名并排序。我们可能还需要找到一种方法，检查“页码”是否发生了变化。如果我们一直这样头痛医头脚痛医脚，很可能会遇到更多的错误。

> 只在脚本上投入你认为必要的时间。如果你想构建一个可持续过程，在很长一段时间内都可以在大型数据集上多次运行，你需要花时间仔细考虑所有步骤。

这就是编程的过程：写代码，调试，写代码，调试。无论是经验多么丰富的程序员，有时都会在代码中遇到错误。在学习编程的过程中，遇到错误会非常沮丧。你可能会想：“为什么无法运行？一定是我不擅长编程。”但事实并非如此。和其他事情一样，编程也需要练习。

现在看来，我们目前的方法显然是行不通的。根据我们目前对文本文件的了解，可以这么说，在利用文本寻找每一部分数据的开始和结束时，我们选用的标志是错误的。我们还可以用这个文件重新开始，换一个角度来思考；但我们想探索解决问题的其他方法，修正错误并获取想要的数据。

5.4 学习解决问题的方法

本节包含好几个练习，你可以试着解析 PDF 脚本，同时挑战自己写 Python 代码的能力。首先，我们先来回顾一下已经写好的代码：

```
pdf_txt = 'en-final-table9.txt'
openfile = open(pdf_txt, "r")

double_lined_countries = [
    'Bolivia (Plurinational \n',
    'Democratic People\xe2\x80\x99s \n',
    'Democratic Republic \n',
    'Lao People\xe2\x80\x99s Democratic \n',
    'Micronesia (Federated \n',
    'Saint Vincent and \n',
    'The former Yugoslav \n',
    'United Republic \n',
    'Venezuela (Bolivarian \n',
]


def turn_on_off(line, status, prev_line, start, end='\n', count=0):
    """
        该函数用于检查该行会是否开始/结束于特定值。
        如果是,且上一行不是特殊行,
        则状态设为开/关(真/假)。
    """
    if line.startswith(start):
        status = True
    elif status:
        if line == end and prev_line != 'and areas':
            status = False
    return status


def clean(line):
    """
       清洗代码行中的换行符、空格以及特殊符号。

    """
    line = line.strip('\n').strip()
    line = line.replace('\xe2\x80\x93', '-')
    line = line.replace('\xe2\x80\x99', '\'')

    return line

countries = []
totals = []
country_line = total_line = False
previous_line = ''

for line in openfile:
    if country_line:
        if previous_line in double_lined_countries:
            line = ' '.join([clean(previous_line), clean(line)])
            countries.append(line)
        elif line not in double_lined_countries:
            countries.append(clean(line))

    elif total_line:
        if len(line.replace('\n', '').strip()) > 0:
```

```
            totals.append(clean(line))

    country_line = turn_on_off(line, country_line,
                               'and areas', previous_line)
    total_line = turn_on_off(line, total_line,
                             'total', previous_line)
    previous_line = line


import pprint
data = dict(zip(countries, totals))
pprint.pprint(data)
```

有好几种方法可以解决我们面临的问题。在接下来的几节中，我们会讲到其中几种解决方法。

5.4.1　练习：使用表格提取，换用另一个库

前面我们对 PDF 转换成文本遇到的困难头痛不已，下面我们寻找其他方法来实现表格提取，不用 `pdfminer`。我们找到了 `pdftables` 库（http://pdftables.readthedocs.org/），这个库已经不再更新了（原作者的最后一次更新时间是两年多以前）。

我们需要安装必要的库（http://pdftables.readthedocs.io/en/latest/#installation），只需运行 `pip install pdftables` 和 `pip install requests` 即可完成安装。原作者并没有及时更新所有的文档，所以文档和 README.md 中的某些例子明显是错的。尽管如此，我们还是找到了一个“多合一”（all in one）的函数，可以用来获取我们想要的数据：

```
from pdftables import get_tables

all_tables = get_tables(open('EN-FINAL Table 9.pdf', 'rb'))

print all_tables
```

我们创建一个新的代码文件（pdf_table_data.py）并运行。你应该会看到旋风般滚动的数据，看起来就是我们要提取的数据。你会注意到，标题并不是完全正确，但每一行的内容似乎都包含在 `all_tables` 变量中。我们来仔细观察一下，看看如何提取我们想要的标题、数据列和注释。

你可能也注意到了，`all_tables` 是一个由列表组成的列表（或者叫*矩阵*）。它有很多行，每一行里还包含很多行。这种方法可能很适合表格提取，因为表格本质上就是行和列。`get_tables` 函数返回的是每一页内容组成的表格，每个表格都包含一个行列表，每个元素又是由许多列组成的列表。

第一步，找到每一列的标题。我们试着查看输出的前几行，看能否找到列标题：

```
print all_tables[0][:6]
```

我们看一下第一页的前六行：

```
... [u'',
  u'',
```

```
  u'',
  u'',
  u'',
  u'',
  u'Birth',
  u'Female',
  u'genital mutila',
  u'tion/cutting (%)+',
  u'Jus',
  u'tification of',
  u'',
  u'',
  u'E'],
 [u'',
  u'',
  u'Child labour (%',
  u')+',
  u'Child m',
  u'arriage (%)',
  u'registration',
  u'',
  u'2002\u201320',
  u'12*',
  u'wife',
  u'beating (%)',
  u'',
  u'Violent disciplin',
  u'e (%)+ 9'],
 [u'Countries  and areas',
  u'total',
  u'2005\u20132012*male',
  u'female',
  u'2005married by 15',
  u'\u20132012*married by 18',
  u'(%)+ 2005\u20132012*total',
  u'prwomena',
  u'evalencegirlsb',
  u'attitudessupport for thepracticec',
  u'2male',
  u'005\u20132012*female',
  u'total',
  u'2005\u20132012*male',
  u'female'],...
```

可以看到，标题都包含在前三个列表中，格式混乱。从 print 语句的输出中还可以看出，每行数据还是相当干净的。如果我们对比 PDF 文件手动设置标题（如下所示），可以得到干净的数据集：

```
headers = ['Country', 'Child Labor 2005-2012 (%) total',
           'Child Labor 2005-2012 (%) male',
           'Child Labor 2005-2012 (%) female',
           'Child Marriage 2005-2012 (%) married by 15',
           'Child Marriage 2005-2012 (%) married by 18',
           'Birth registration 2005-2012 (%)',
```

```
            'Female Genital mutilation 2002-2012 (prevalence), women',
            'Female Genital mutilation 2002-2012 (prevalence), girls',
            'Female Genital mutilation 2002-2012 (support)',
            'Justification of wife beating 2005-2012 (%) male',
            'Justification of wife beating 2005-2012 (%) female',
            'Violent discipline 2005-2012 (%) total',
            'Violent discipline 2005-2012 (%) male',
            'Violent discipline 2005-2012 (%) female']  ❶

for table in all_tables:
    for row in table:
        print zip(headers, row)  ❷
```

❶ 将所有标题添加到一个列表中，其中包括国名。现在我们可以将这个列表与行数据合并，将数据和标题对齐。
❷ 使用 zip 方法将标题与每一行数据合并。

从代码输出中可以看出，有些行我们已经匹配好了，但还有很多行不是国家行（和我们之前的结果类似，之前在表格中发现了多余的空格和空行）。

根据目前所学的内容，我们希望用编程加测试的方法解决这个问题。我们知道有些国家占了不止一行。我们还知道 PDF 文件用破折号（-）表示数据缺失，所以全空的行实际上不是数据行。从上一次 print 输出中我们就知道，每一页的数据从第五行开始。我们还知道，我们关注的最后一行是津巴布韦（Zimbabwe）。将我们已知的内容综合在一起，我们得到：

```
for table in all_tables:
    for row in table[5:]: ❶
        if row[2] == '':  ❷
            print row
```

❶ 在每一页中找出我们想要的那些行，即索引数为 5 之后的切片。
❷ 如果数据为空，打印查看该行内容。

运行代码，你会发现列表中包含一些随机分布的空白行，其中也不包含国名。这可能就是我们上一段脚本的问题所在。我们尝试将国名合并在一起，跳过其他空白行。我们还加上对津巴布韦（Zimbabwe）的测试：

```
first_name = ''

for table in all_tables:
    for row in table[5:]:
        if row[0] == '':  ❶
            continue
        if row[2] == '':
            first_name = row[0]  ❷
            continue
        if row[0].startswith(' '):  ❸
            row[0] = '{} {}'.format(first_name, row[0])
        print zip(headers, row)  ❹
        if row[0] == 'Zimbabwe':
            break  ❺
```

❶ 如果数据行索引数为 0 的值缺失，说明这一行不包含国名，是一个空行。下一行代码用 continue 跳过这一行，continue 是一个 Python 关键字，作用是转到 for 循环的下一次迭代。
❷ 如果数据行索引数为 2 的值缺失，我们知道这可能是国名的前半部分。本行代码将国名的前半部分保存为变量 first_name。下一行代码跳转到下一行数据。
❸ 如果数据行以空格开头，我们知道这是国名的后半部分。我们希望将国名的两部分重新合并在一起。
❹ 如果我们的假设正确，观察打印出的结果，数据应该是匹配好的。本行代码打印出每次迭代的内容，便于我们观察。
❺ 遇到津巴布韦（Zimbabwe）时，本行代码跳出 for 循环。

大部分数据看起来都是正确的，但我们还会发现一些异常数据。看下面这个例子：

```
[('Country', u'80      THE STATE OF T'),
('Child Labor 2005-2012 (%) total', u'HE WOR'),
('Child Labor 2005-2012 (%) male', u'LD\u2019S CHILDRE'),
('Child Labor 2005-2012 (%) female', u'N 2014'),
('Child Marriage 2005-2012 (%) married by 15', u'IN NUMBER'),
('Child Marriage 2005-2012 (%) married by 18', u'S'),
('Birth registration 2005-2012 (%)', u''),
.....
```

可以看到，开头的页码被误以为是国名。你知道哪些国家名称里有数字吗？我们当然不知道！我们添加一个对数字的测试，看是否能剔除坏数据。我们还注意到双行国家的对应并不正确。从输出来看，pdftables 在导入数据时会自动修正行首的空格。太好了！现在我们应该添加一个测试，测试上一行数据有没有 first_name：

```
from pdftables import get_tables
import pprint

headers = ['Country', 'Child Labor 2005-2012 (%) total',
           'Child Labor 2005-2012 (%) male',
           'Child Labor 2005-2012 (%) female',
           'Child Marriage 2005-2012 (%) married by 15',
           'Child Marriage 2005-2012 (%) married by 18',
           'Birth registration 2005-2012 (%)',
           'Female Genital mutilation 2002-2012 (prevalence), women',
           'Female Genital mutilation 2002-2012 (prevalence), girls',
           'Female Genital mutilation 2002-2012 (support)',
           'Justification of wife beating 2005-2012 (%) male',
           'Justification of wife beating 2005-2012 (%) female',
           'Violent discipline 2005-2012 (%) total',
           'Violent discipline 2005-2012 (%) male',
           'Violent discipline 2005-2012 (%) female']

all_tables = get_tables(open('EN-FINAL Table 9.pdf', 'rb'))

first_name = False
final_data = []

for table in all_tables:
    for row in table[5:]:
        if row[0] == '' or row[0][0].isdigit():
```

```
            continue
        elif row[2] == '':
            first_name = row[0]
            continue
        if first_name:  ❶
            row[0] = u'{} {}'.format(first_name, row[0])
            first_name = False  ❷
        final_data.append(dict(zip(headers, row)))
        if row[0] == 'Zimbabwe':
            break

pprint.pprint(final_data)
```

❶ 如果这一行有 `first_name`，那么在该行内将国名合并。

❷ 将 `first_name` 重新设置为 `False`，这样下一次迭代可以正常运行。

现在数据导入工作已全部完成。如果你希望数据结构与从 Excel 导入的数据完全相同，需要对数据做进一步处理，但我们已经可以将 PDF 中的数据保存成行数据。

> `pdftables` 已经不再受到积极的支持，它的开发者现在提供替代的新产品，但却是收费的（https://pdftables.com/）。依赖不受支持的代码是很危险的，我们也不能认为 `pdftables` 总是可用[3]。但是，开源社区的一部分内容就是回馈，所以我们鼓励你找到好项目，为它做贡献，帮它宣传，希望像 `pdftables` 这样的项目能够保持开源，能够继续成长并发展。

下面，我们来看一下解析 PDF 数据的其他方法，其中包括手动清洗数据。

5.4.2　练习：手动清洗数据

我们来聊一聊一个大家闭口不谈却确实存在的事实。阅读本章的过程中，你可能一直想知道，我们为什么不修改 PDF 文本文件，这样处理起来会更方便。你可以这么做，这是解决问题的众多方法之一。但我们希望你能挑战一下，用 Python 工具处理这个文件。你也不是每次都能手动修改 PDF 文件。

如果在处理 PDF 或其他文件类型时遇到了困难，一种按部就班的方法是将数据提取到文本文件，然后手动处理数据。在这种情况下，提前预估一下你愿意在手动处理上花费的时间，然后将实际花费的时间控制在这个范围内。

想了解数据清洗自动化的更多内容，请查阅第 8 章。

5.4.3　练习：试用另一种工具

当最开始寻找用来解析 PDF 的 Python 库时，我们在网络上搜索其他人如何完成这个任务，并找到了 `slate`，它看起来很好用，但需要一些自定义代码。

注 3：似乎的确有人在维护并支持一些活跃的 GitHub 分支（https://github.com/drj11/pdftables/network）。我们建议你关注这些仓库的动态，以满足 PDF 表格解析的需求。

想了解还有哪些可用的工具，我们试着搜索“extracting tables from pdf”（从 pdf 中提取表格），而不是搜索“parsing pdf python”（解析 pdf python），这样可以找到针对表格问题的解决方法（其中有一篇博客文章对几种工具做了对比，http://www.interhacktives.com/2014/03/12/extract-data-pdf/）。

对于像我们要解析的这种小型 PDF，我们可以使用 Tabula（http://tabula.technology/）。Tabula 不一定总能解决问题，但它有一些很好的功能。

Tabula 的使用方法如下。

(1) 下载 Tabula（http://tabula.technology/）。
(2) 双击启动应用，这会在浏览器中打开 Tabula 工具。
(3) 上传童工 PDF 文件。

从这里开始，你需要修改 Tabula 选择抓取的内容。跳过标题行可以让 Tabula 找到每一页的数据并自动高亮，方便后续提取。首先，选择你感兴趣的表格（见图 5-3）。

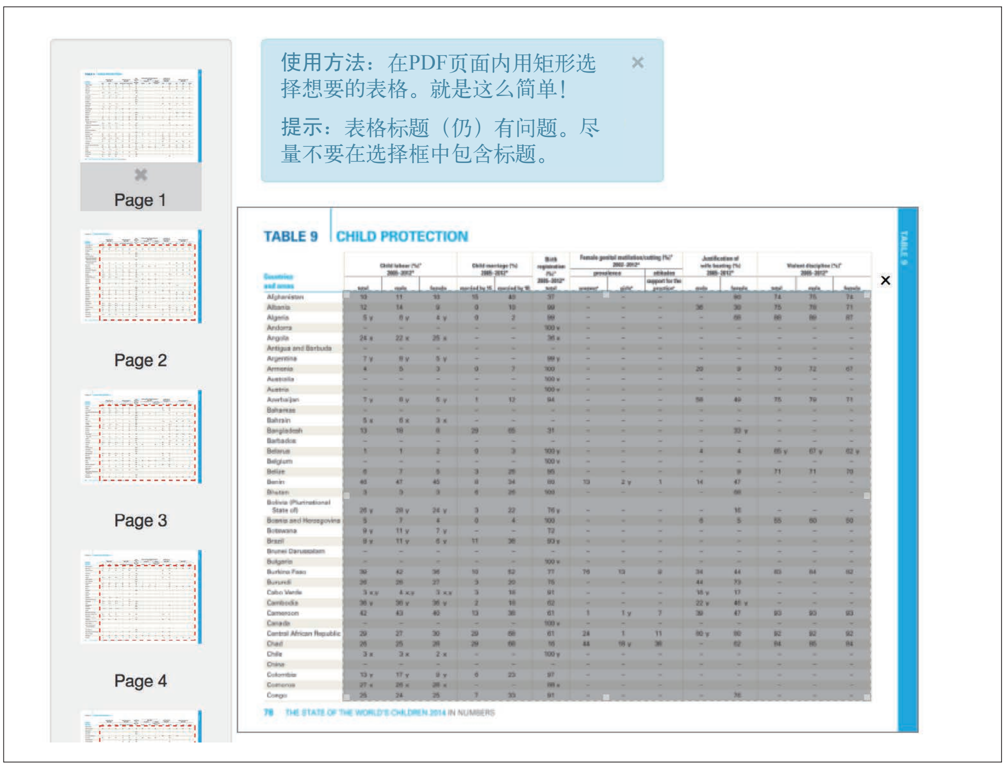
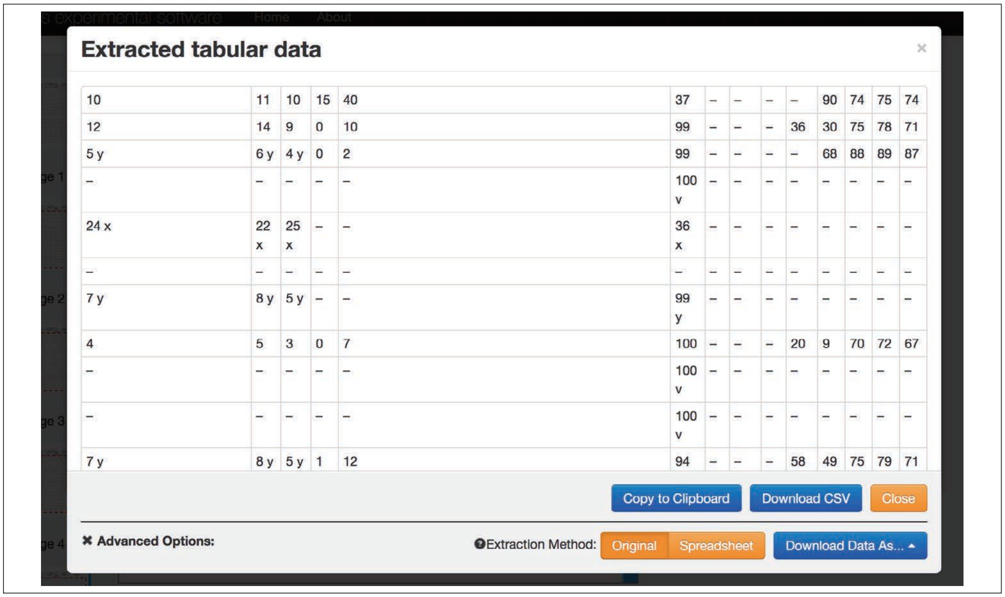


图 5-3：在 Tabula 中选择表格

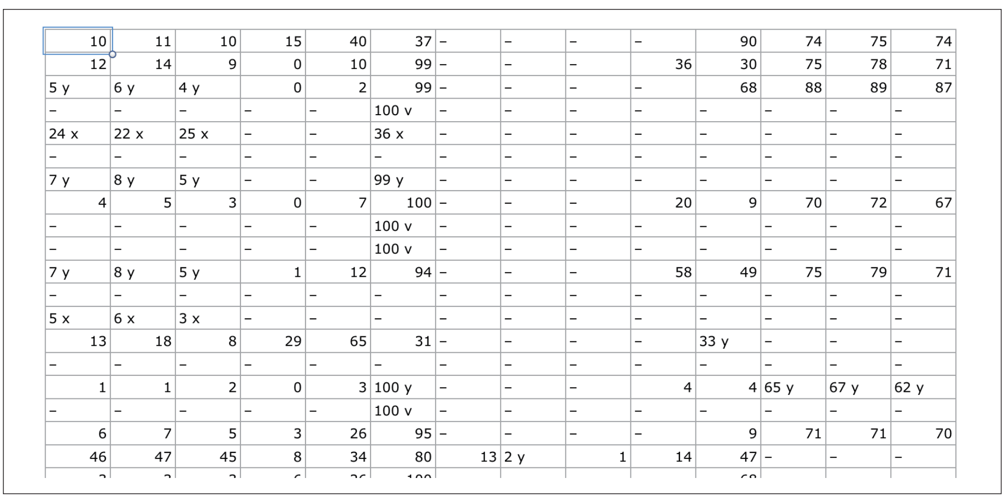
接下来，下载数据（见图 5-4）。



10	11	10	15	40	37	–	–	–	–	90	74	75	74
12	14	9	0	10	99	–	–	–	36	30	75	78	71
5 y	6 y	4 y	0	2	99	–	–	–	–	68	88	89	87
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
24 x	22 x	25 x	–	–	36 x	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–
7 y	8 y	5 y	–	–	99 y	–	–	–	–	–	–	–	–
4	5	3	0	7	100	–	–	–	20	9	70	72	67
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
7 y	8 y	5 y	1	12	94	–	–	–	58	49	75	79	71

图 5-4：Tabula 的下载界面

点击“Download CSV”（下载 CSV 文件），你会得到类似图 5-5 中的数据。



10	11	10	15	40	37	–	–	–	–	90	74	75	74
12	14	9	0	10	99	–	–	–	36	30	75	78	71
5 y	6 y	4 y	0	2	99	–	–	–	–	68	88	89	87
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
24 x	22 x	25 x	–	–	36 x	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–
7 y	8 y	5 y	–	–	99 y	–	–	–	–	–	–	–	–
4	5	3	0	7	100	–	–	–	20	9	70	72	67
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
7 y	8 y	5 y	1	12	94	–	–	–	58	49	75	79	71
–	–	–	–	–	–	–	–	–	–	–	–	–	–
5 x	6 x	3 x	–	–	–	–	–	–	–	–	–	–	–
13	18	8	29	65	31	–	–	–	–	33 y	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–
1	1	2	0	3	100 y	–	–	–	4	4	65 y	67 y	62 y
–	–	–	–	–	100 v	–	–	–	–	–	–	–	–
6	7	5	3	26	95	–	–	–	–	9	71	71	70
46	47	45	8	34	80	13	2 y	1	14	47	–	–	–

图 5-5：提取的 CSV 数据

得到的数据并不完美，但比我们用 `pdfminer` 得到的数据更干净。

接下来的挑战是，解析 Tabula 创建的 CSV 文件。这与我们解析过的其他 CSV（见第 3 章）有所不同，要更杂乱一些。如果你遇到困难，可以先放在一边，等读完第 7 章再回来解决它。

5.5　不常见的文件类型

目前为止，本书已经讲过 CSV、JSON、XML、Excel 和 PDF 文件。PDF 中的数据很难解析，你可能认为数据解析的世界不能比这更糟了。遗憾的是，还有比这更糟糕的事情。

好消息是，你可能不会遇到前人尚未解决的问题。记住，向 Python 社区或更高一级的开源社区寻求帮助和建议，这永远都是一个好方法，即使你已经认识到应该寻找更容易解析的数据集。

如果数据具有以下特征，你可能会遇到问题。

- 文件由旧系统生成，使用的是一种不常见的文件类型。
- 文件由专用系统（proprietary system）生成。
- 你所有的程序都无法打开该文件。

对于与不常见文件类型相关的问题，仅仅用你之前学过的知识就可以解决。

(1) 确定文件类型。如果从文件扩展名上不容易看出，那么可以用 `python-magic` 库（https://pypi.python.org/pypi/python-magic/0.4.6）。

(2) 在互联网上搜索“how to parse <file extension> in Python”（用 Python 如何解析 < 文件扩展名 >），将“<file extension>”替换为实际的文件扩展名。

(3) 如果找不到显而易见的解决方法，尝试用文本编辑器打开该文件，或者用 Python 的 `open` 函数读取该文件。

(4) 如果字符看起来很奇怪，读一些关于 Python 编码的内容。如果你是第一次接触 Python 字符编码，可以观看 PyCon 2014 的演讲“Python 中的字符编码和 Unicode”（https://www.youtube.com/watch?v=Mx70n1dL534）。

5.6　小结

PDF 以及其他难以解析的格式，是你会遇到的最糟糕的格式。如果你找到这些格式的数据，应该做的第一件事就是看能否找到其他格式的数据。对于我们的例子来说，我们从 CSV 格式得到的数据更为精确，因为 PDF 表格中的数字是经过四舍五入的。越是原始的数据格式，数据可能就越精确，用代码解析也越容易。

如果找不到其他格式的数据，那你应该尝试以下步骤。

(1) 确定数据类型。

(2) 在互联网上搜索其他人解决问题的方法。有没有帮助导入数据的工具？

(3) 凭直觉选择你要用的工具。如果是 Python，选择你认为最合适的库。

(4) 尝试将数据转换成更容易使用的格式。

表 5-1 列出了我们在本章学过的库和工具。

表5-1：新的Python库和工具

库或工具	作用
slate	每次运行脚本时，都将 PDF 解析为内存里的一个字符串
pdfminer	将 PDF 转换为文本，这样你就可以解析文本文件
pdftables	首先用 pdfminer 将 PDF 解析成文本，然后尝试寻找表格，并将每一行内容对齐
Tabula	提供操作界面，可以将 PDF 数据提取成 CSV 格式

除了上面这些新工具，我们还学习了 Python 编程的一些新概念，表 5-2 对这些新概念做了汇总。

表5-2：Python编程新概念

概念	作用
转义字符（http://learnpythonthehardway.org/book/ex10.html）	转义字符告诉计算机，文件路径或文件名中有一个空格或特殊字符，告知方式是在其前面加一个反斜线（\）。一种用法是在空格前加 \ 将其转义
\n	\n 是文件中行尾或新行的标志
elif（https://docs.python.org/2/tutorial/controlflow.html）	在写 if-else 语句的过程中，我们可以添加额外的条件再次测试：if...elif...elif...else
函数（https://docs.python.org/2/tutorial/controlflow.html#defining-functions）	Python 函数用来执行一段代码。将可复用的代码写成函数，我们可以避免多次重复
zip（https://docs.python.org/2.7/library/functions.html#zip）	zip 是 Python 内置函数，将两个可迭代对象转换成由元组构成的列表
元组（https://docs.python.org/2.7/library/functions.html#tuple）	元组和列表类似，但是不可更改（immutable），也就是说，不能修改元组。想修改一个元组，需要将其保存为一个新对象
dict 转换（https://docs.python.org/2.7/library/functions.html#func-dict）	dict 是 Python 内置函数，将输入转换成字典。输入数据应满足键值对的形式，这样函数才能正常运行

下一章我们将讨论数据获取与存储。这样我们就能了解关于获取其他数据格式的更多内容。在第 7 章和第 8 章，我们会讲到数据清洗，这也对我们处理复杂的 PDF 有所帮助。

第 6 章 数据获取与存储

找到要研究的第一个数据集，可能是向回答问题这一目标迈出的最重要一步。在第 1 章里我们说过，你首先应该花点时间将问题细化，让问题足够具体，能够找到关于问题的好数据，同时问题又要足够宽泛，可以让你和其他人都感兴趣。

另一种可能是，你已经找到了感兴趣的数据集，但无法提出令人信服的问题。如果你不了解也不信任数据来源，应该花点时间调查一下。问问你自己：数据是否有效？是否更新过？我能否信任当前以及未来的更新和出版物？

本章我们会讲到，你可以将数据保存在什么地方，以供后续使用。如果你不熟悉数据库的话，我们也会讲到数据库的使用场景和使用方法，并演示如何创建简单数据库来存储数据。如果你已经很熟悉数据库，或者你的数据源就是一个数据库的话，我们会讲到 Python 中基本的数据库连接结构。

如果你还没决定使用哪个数据集的话，不必担心。下面用的几个例子，你都可以在本书仓库（https://github.com/jackiekazil/data-wrangling）中找到。

> 我们强烈建议你带着几个问题阅读本书，这样你才能更好地在实践中学习。这些问题可能是你一直想研究的问题，也可能是与本书所探索数据相关的问题。即使你选取的问题很简单，在编写代码中学习也是最好的学习方法。

6.1 并非所有数据生而平等

对于遇到的每一个数据集，尽管我们愿意相信其真实性和数据质量，但并非所有数据都能符合我们的预期。即使是你目前使用的数据集，在深入研究之后也可能是无用且无效的数据源。对于你面临的数据处理问题，在寻求自动化解决方案时，你会发现 Python 工具可

以帮你分辨好数据和坏数据，还可以帮你评价数据的可用性。第 7 章和第 8 章会讲到用 Python 做数据清洗和数据探索，第 14 章会讲到自动化，在这些章节里我们都会介绍关于这些工具的更多内容。

刚刚得到新数据时，我们建议做一个数据*气味测试*，测试该数据是否是可靠的信息源，并决定是否信任该数据。你可以问问自己以下几个问题。

- 如果我有问题或疑虑的话，能够联系上作者本人吗？
- 数据是否定期检查错误并更新？
- 数据里是否包含数据获取方法的信息，是否包含数据获取过程中使用的样本类型？
- 有没有其他数据源可以验证这个数据集？
- 根据我对这个话题了解的所有知识，数据看起来是否可信？

如果你对至少三个问题的回答都是“是”，这说明你走对路了！如果至少对两个问题的回答是“否”，你可能需要花更多时间寻找可靠的数据。

> 你可能需要联系最初采集数据并发布的作者或机构，以寻求更多信息。通常情况下，给合适的人打电话或发电子邮件，可以帮你回答上面至少一个问题，并验证数据源的可靠性。

6.2 真实性核查

为保证报告的可信，对你的数据做真实性核查是非常重要的，尽管有时可能既烦人又累人。根据数据集的不同情况，真实性核查可能包括以下内容。

- 联系数据源，核实最新的方法和版本。
- 找到其他好的数据源作对照。
- 联系专家，探讨好的数据源和真实信息。
- 进一步研究你选定的主题，检查你的数据源和 / 或数据集是否可信。

有些图书馆和大学可以访问只对订阅用户开放的出版物和教育档案，它们是真实性核查的重要资源。如果可以访问类似 LexisNexis（http://lexisnexis.com/）、国会季刊新闻库（http://library.cqpress.com/）、JSTOR（http://jstor.org/）、康奈尔大学的 arXiv 项目（http://arxiv.org/）和谷歌学术搜索（http://scholar.google.com/）这样的工具，你可以看一下其他人对你的主题已经做了哪些研究及研究成果。

谷歌搜索也可以在真实性核查方面提供帮助。如果某人宣称数据来自于公开发布的来源，那么有可能其他人已经对这一说法做过真实性核查，或者已经证实了这一说法。同样，在评价网上发布的内容时，你需要自己仔细斟酌。数据源是否真实？论证能否令人信服，是否有意义？证据是否有效？对这些问题的回答要做综合评价。

> 政府部门拥有大量的数据集。如果想研究本地城市、州或国家的某一现象，通常你可以通过电话或电子邮件找到某个人，他手头有很有用的数据集。全球人口普查局会定期发布普查数据，如果你纠结于想要回答什么问题，可以从这些数据开始。

对第一个数据集做过验证和真实性核查之后，以后编写脚本验证数据有效性就会容易很多。你甚至可以用在本书中学到的技巧（特别是第 14 章的内容）创建脚本来自动更新数据。

6.3　数据可读性、数据清洁度和数据寿命

如果你的数据集看起来非常难以读取，还有一种可能的方法：根据第 7 章学习的内容，你可以用代码清洗数据。幸运的是，如果是计算机创建的数据，很有可能可以被计算机读取。更大的难点在于，从“真实生活”中获取数据并读入计算机。在第 5 章中我们知道，PDF 和不常见的数据文件类型很难处理，但并非不可能。

我们可以用 Python 帮我们读取难以读取的数据，但难以读取可能意味着数据来源不佳。如果是计算机生成的大型数据集，那就存在一个问题——数据库转储（database dump）的格式一直都不美观。但如果你的数据是人工生成的，而且难以读取，那可能是数据清洁度和数据有效性的问题。

你面临的另一个问题是，数据是否已经被清洗过了。通过详细询问数据是如何采集、报告并更新的，你可以判断数据是否被清洗过。你应该能够确定以下内容。

- 数据的清洁度有多高？
- 是否有人给出了统计误差率，或者修改了错误的数据条目，或者误报了数据？
- 是否会发布进一步更新，这些更新是否会发送给你？
- 数据采集过程中使用了哪些方法，如何验证这些方法？



如果你的数据源使用的是标准化的、严谨的研究和采集方法，在未来的几年里，你的清洗脚本和报告脚本可能几乎不用修改就可以重复使用。那些系统通常不会定期发生变化（变化既费钱又费时）。一旦写好了清洗脚本，你可以轻松处理下一年的数据，直接进入数据分析阶段。

除了清洁度和可读性，你还要关注数据的寿命。你要处理的数据是定期采集并更新的吗？数据发布和更新的时间计划是什么样的？了解一个机构更新数据的频率，可以让你判断在未来几年内对该数据的应用能力。

6.4　寻找数据

就像验证数据源或编写 PDF 解析器脚本有不止一种方法一样，寻找数据也有很多种方法。本节会讲到你在线上和线下都可以使用的方法。

6.4.1　打电话

观察数据文件，并思考一个问题：数据是怎么变成现在这种格式的？通常来说，像 Excel、PDF 甚至 Word 这样的文件类型都是人工处理的，这个人从数据源处获取数据。

如果你找到了采集数据的那个人，或许可以要到原始数据。原始数据可能是易于解析的文件格式，如 CSV 或数据库。你找到的那个人还可以回答关于采集方法和更新时间线的问题。

下面给出从数据文件中找人的一些技巧。

- 在文件中搜索联系人信息。
- 寻找署名——如果没有人名，那就寻找机构名。
- 在网络上搜索文档的文件名和标题。
- 右键单击文件，在 Windows 上选择“属性”（在 Mac 上选择“显示简介”），查看文件元数据。

去联系你能找到的每一个人。如果他不是创建文件的人，可以问他知不知道是谁创建的文件。不要害羞——你对他们的研究课题和工作感兴趣，就是对他们的恭维，他们会很乐意帮助你的。

与通信官打交道

如果你遇到这种情况——发布文件的机构希望你能和他们的通信代表谈一谈——这意味着时间可能会拖得很长。还记得一个叫作打电话的游戏吗：第一个人跟另一个人说了些什么，另一个人将所听到的内容复述给下一个人，如此这般，最后一个人的话已经与第一个人大相径庭？

要保证有效沟通，你可以做这两件事情。第一，努力建立信任。如果没有利益冲突，你可以分享你感兴趣的工作，并承诺会将该机构列为数据源。这表示你会间接宣传他们的工作，该机构也会在分享资料方面受到好评。第二，请求通信代表召开电话会议或有监督的讨论。通过电话而不是电子邮件沟通，你可以及时准确地得到问题的回答。

找到了要联系的人之后，尝试用电话联系他，或者亲自拜访。电子邮件很容易引起误会，通常时间也会拖得比较长。下面给出几个问题的例子，可以帮你思考要问什么样的问题。

- 你是如何获取第 6 页到第 200 页的数据的？
- 是否有其他格式的数据，比如 JSON、CSV、XML 或数据库？
- 数据是如何采集的？
- 能否描述一下数据采集的方法？
- 这些缩写是什么意思？
- 数据是否会更新？如何更新？何时更新？
- 是否有其他人能提供更多信息？

在等待回答的同时，你也可以开始做数据探索，这取决于项目的时间限制和目标。

6.4.2 美国政府数据

如果你对研究美国的现象感兴趣，奥巴马政府最近正在推行发布易于获取的在线数据，可以很容易找到政府机构的定期报告。快速浏览一下 Data.gov 网站，你就会发现暴雨数据（http://catalog.data.gov/dataset/ncdc-storm-events-database）、毕业率和辍学率（http://catalog.data.gov/dataset/edfacts-graduates-and-dropouts-201112）、濒危物种数据（http://catalog.data.gov/dataset/density-of-threatened-and-endangered-species）、犯罪统计（https://catalog.data.gov/dataset/total-crime-index-for-the-nations-largest-cities-3a1aa）以及其他有趣的数据集。

除了联邦数据，州政府和地方政府也都有发布数据的网站，下面我们列出了其中几个：

- 教育数据（http://datainventory.ed.gov/InventoryList）
- 选举结果（http://www.fec.gov/pubrec/electionresults.shtml）
- 人口普查数据（http://census.ire.org/）
- 环境数据（https://www.epa.gov/enviro/about-data）
- 劳工统计数据（http://www.bls.gov/）

如果在公开发布的信息里找不到你想要的，不要犹豫，直接给相应部门打电话，在电话里请求提供数据。许多政府部门都有实习生或工作人员负责处理公众获取信息的请求。

> **信息自由法案介绍**
>
> 在美国，你可以向任意地方、州或联邦的政府机关提交信息自由法案（FOIA）申请。申请应简单明了。根据你请求的信息内容以及描述的详细程度，需要用的时间可能会有很大不同。
>
> 美国政府建立了 FOIA 网站（https://foiaonline.regulations.gov/foia/action/public/home），你可以在上面向特定部门提交申请并追踪，但大部分政府机构都有在自己的网站上如何提交 FOIA 申请的说明。你应该在申请中给出联系信息，描述你要找的数据记录，以及如果有复印费的话你愿意交多少钱。
>
> 好的做法是，尽可能详细描述你要找的数据记录，但又不过分限制搜索的范围。你可以这么想，搜索范围太宽泛，网站会返回数百万条记录（你需要手动挑选，可能还要付费）。另一方面，如果搜索过于具体，你可能会漏掉与话题密切相关的记录。当然了，根据你第一次申请找到的信息，你总可以提交更多的 FOIA 申请。搜索过程也很有意思，不是么？
>
> 如果你想要的是美国之外政府和机构的信息，维基百科给出了世界各国信息自由法律的列表（https://en.wikipedia.org/wiki/Freedom_of_information_laws_by_country）。想了解关于美国 FOIA 的更多内容，可参阅电子前沿基金会（Electronic Frontier Foundation）的建议（https://www.eff.org/issues/transparency/foia-how-to）。

6.4.3　全球政府和城市开放数据

获取政府数据有很多种方法，这取决于你想研究的国家，以及你是否生活在那个国家。由于我们更熟悉美国的政策，所以我们不会宣称这是一份全面的清单。如果你想要分享本书没有提到而又很有用的开放数据，请随时和我们联系！

> 我们建议对政府数据集也要做真实性核查，特别是对有侵犯人权历史的政府更要仔细核查。将所有的判断力用到所有数据上，果断给联系人打电话或发邮件，进一步咨询数据的采集方法。

1. 欧盟和英国

如果你对欧盟或英国的数据感兴趣，可以找到许多数据门户网站。下面一些网站是由一些机构和开放数据爱好者创建的，如果你想寻找特定的数据集，可以和网站所有者直接联系。

- 欧盟开放数据（http://publicdata.eu/）
- 欧罗巴开放数据（http://open-data.europa.eu/）
- 全天开放关联数据（http://latc-project.eu/）
- 英国政府数据（https://data.gov.uk/）

2. 非洲

如果你对非洲国家的数据感兴趣，有许多项目正在采集数据并构建 API，供开发人员使用。许多非洲国家也有自己的开放数据门户网站（用谷歌一搜就可以找到）。我们挑出了一些有用的区域性项目：

- 非洲开放数据（https://africaopendata.org/）
- 南非代码（http://code4sa.org/）
- 非洲代码（https://codeforafrica.org/）
- 非洲的开放数据（http://opendataforafrica.org/）

3. 亚洲

如果你对亚洲国家和地区的数据感兴趣，它们大多数都有自己的开放数据网站。我们找出了几个令人印象深刻的数据集，以及一些机构发布的区域性数据：

- 开放城市项目（http://www.opencitiesproject.org/）
- 开放尼泊尔（http://data.opennepal.net/）
- 中国国家统计局（http://www.stats.gov.cn/english/）
- 香港开放数据（https://opendatahk.com/）
- 印尼政府开放数据（http://data.go.id/）

4. 欧盟以外的欧洲、中亚、印度、中东和俄罗斯

在欧盟之外，许多中亚、中欧和中东的国家也有自己的政府开放数据网站。我们给出了其中一些网站，但如果你知道你想研究的国家和地区并希望用母语来访问相关数据，语言技能是最重要的（谷歌 Chrome 浏览器会尝试自动翻译网页，所以即使语言不通也可以找到有用的数据）。

- 俄罗斯政府数据网站（http://data.gov.ru/）
- PakReport——巴基斯坦开放数据和地图（http://pakreport.org/）
- 印度开放数据（https://data.gov.in/）
- 土耳其开放统计数据（http://www.turkstat.gov.tr/Start.do）

5. 南美和加拿大

许多南美国家都有自己的开放数据网站，通过搜索很容易找到。加拿大也有针对统计数据的开放数据门户网站。我们给出了其中一些网站，同时建议你去网上搜索，寻找你感兴趣的特定部门或政府。

- 加拿大统计数据（http://www.rdc-cdr.ca/datasets-and-surveys）
- 加拿大开放数据（http://open.canada.ca/en）
- 巴西开放数据（http://dados.gov.br/）
- 墨西哥开放数据（http://datos.gob.mx/）

- 拉丁美洲开放数据（http://www.opendatalatinoamerica.org/）
- 发展中的加勒比地区（https://www.developingcaribbean.org/#/）

6.4.4 组织数据和非政府组织数据

无论是地方组织还是国际组织，都有大量跨州或跨国的数据集资源，比如气候变化数据、国际商贸数据和全球运输数据。如果政府并没有采集与你的主题相关的数据（关于宗教细节、吸毒、社区支持网络等的数据），或者政府数据不可靠，或者政府没有开放数据门户网站的话，你可以通过 NGO 或开放数据组织找到相关数据。下面列出了一些组织，但还有更多的组织在为数据的公开交换和访问而奋斗。

- 联合国开放数据（http://data.un.org/）
- 联合国发展计划署数据（http://open.undp.org/）
- 开放知识基金会（https://okfn.org/）
- 世界银行数据（http://data.worldbank.org/）
- 维基解密（https://wikileaks.org/）
- 国际援助透明度数据集（http://www.iatiregistry.org/）
- DataHub（https://datahub.io/）
- 人口资料局（http://www.prb.org/DataFinder.aspx）

6.4.5 教育数据和大学数据

世界各地的大学和研究生部都在不断地研究并发布数据集，从生物科学的进展到本土文化与周边生态栖息地的关联性，涵盖范围很广。很难想象教育领域还没有讨论过某一主题，所以大学是获取最新专题数据的好去处。大多数研究者都乐于听到有人对他们的课题感兴趣，所以我们建议你直接联系合适的部门或作者，以获取更多信息。如果你不知道从哪里开始，下面有几个不错的选择。

- Lexis Nexis（http://www.lexisnexis.com/）
- 谷歌学术搜索（https://scholar.google.com/）
- 康奈尔大学 arXiv 项目（http://arxiv.org/）
- UCI 机器学习数据集（http://archive.ics.uci.edu/ml/）
- 通用数据集倡议（http://www.commondataset.org/）

6.4.6 医学数据和科学数据

与大学类似，科学和医学研究部门和组织也都拥有大量优质的数据资源。在科学研究中搜索是十分困难的，但不要气馁——如果你能找到用于研究的数据集，它们使用的研究术语往往并不相同。如果你想到某一个特定的研究者，我们建议直接联系他。下面列出了一些汇总的数据集：

- 开放科学数据云（https://www.opensciencedatacloud.org/publicdata/）
- 开放科学目录（http://www.opensciencedirectory.net/）
- 世界卫生组织数据（http://www.who.int/gho/database/en/）

- Broad 研究所开放数据（http://www.broadinstitute.org/scientific-community/data）
- 人类连接组项目（神经通路映射）（http://www.humanconnectomeproject.org/）
- UNC 精神病基因组协会（http://www.med.unc.edu/pgc/）
- 社会科学数据集（http://3stages.org/idata/）
- CDC 医学数据（http://www.cdc.gov/nchs/fastats/）

6.4.7 众包数据和API

如果你的想法或问题更适合众包，则可以利用互联网及大量的论坛、服务和社交媒体来创建自己的问题，并用数据挖掘方法找到这些问题的答案。像 Twitter 和 Instagram 这样的服务拥有数亿用户，上面还有好用的应用编程接口（API）。API 是一些协议或工具，允许用软件或代码与另一个系统交互。在我们的例子中，我们使用的一般是基于网络的 API，可以发送网络请求并从服务中获取数据。一般来说，不到一个小时的设置，API 访问就可以获取数百万条数据记录。

在第 13 章我们会更深入地介绍 API，现在，我们在表 6-1 中对比了使用 API 的一些基本优点和缺点。

表6-1：使用API

优点	缺点
即时访问可用的数据	大量 API 系统不可靠（选择性偏差）
数据量很大	数据过载
你不必担心存储问题，你可从服务的存储中访问数据	可靠性问题，依赖于 API 访问限制或停机时间

可以看到，API 的优点和缺点各占一半。如果找到一个你想用的 API，你可以制定一些规则，规定如何使用 API，以及 API 无法访问时应该怎么做（你可能希望把响应内容保存在本地，避免遇到停机问题）。长时间对响应内容进行采集，也可以消除研究中的一些选择性偏差。

除了社交网络服务之外，还有许多网站可以发布你的问题和想法，以寻求众包回答。选择与话题相关的专家论坛，还是自己发布调查并利用自己的频道传播，这由你自己决定，但如果用的是你自己的研究问题和方法，你一定要对样本大小和样本误差作出解释。想要做附带详细引文的抽样调查，更详细的介绍内容可优先参考威斯康星大学的调查指南（http://oqi.wisc.edu/resourcelibrary/uploads/resources/Survey_Guide.pdf）。

想了解其他方面的众包数据，可查看：

- 盖洛普民意调查（http://www.gallup.com/）
- 欧洲社会调查（http://www.europeansocialsurvey.org/data）
- 路透社民意调查（http://polling.reuters.com/）

可用的数据量是巨大的，在大量的噪声数据中，找出你能回答的问题并搞清楚应该如何回答这些问题，可不是一件容易的事情。下面我们讲几个案例研究，让你更好地了解如何寻找数据来回答自己的问题。

6.5　案例研究：数据调查实例

我们将简单介绍几个不同的兴趣领域和问题，这样你可以知道第一步该做些什么。

6.5.1　埃博拉病毒危机

比方说，你对调查西非的埃博拉病毒危机感兴趣。你会怎么开始调查？你可能很快会想到用谷歌搜索“Ebola crisis data”（埃博拉病毒危机数据）。你发现有许多国际组织致力于追踪病毒的传播，这些组织提供了许多工具，任你使用。首先，你会找到 WHO 的情况报告（http://apps.who.int/ebola/ebola-situation-reports）。WHO 网站上有关于最新病例和死亡的信息，还有交互式地图显示受影响的地区，以及应对措施的关键绩效指标，这些内容似乎都是每周更新。数据有 CSV 和 JSON 两种格式，是真实可靠、定期更新的信息来源。

你要不断挖掘寻找其他可用的资源，而不是在出现的第一个结果这里就止步不前。经过进一步搜索，我们找到 GitHub 用户 cmrivers 的仓库（https://github.com/cmrivers/ebola），里面是来自许多政府和媒体数据源的原始数据汇总。由于我们知道该用户，可以通过联系方式联系到他们，所以我们还可以核实数据最近一次的更新时间，并咨询任何与数据采集方法有关的问题。我们学过如何处理这些数据格式（CSV、PDF 文件），所以处理起来应该不成问题。

进一步深入挖掘，你可能会专注于一个具体的问题，比如：“在安全下葬方面采取了哪些预防措施？”你找到一份由 Sam Libby（https://data.humdata.org/user/libbys）维护的报告，报告内容是关于安全、庄严的葬礼的（https://data.humdata.org/dataset/safe-and-dignified-burial-teams）。太棒了！遇到任何问题你都可以直接联系 Sam。

你已经找到了一系列很好的初始数据源，并确认它们来自你信任的组织，同时还找到了联系人，在研究过程中可以向他寻求更多信息。下面我们来看另一个例子。

6.5.2　列车安全

再比方说，你对美国的列车安全感兴趣。你的问题可能是：有哪些影响列车安全的不利因素？首先，你可能看过之前关于列车安全的研究。你找到了联邦铁路管理局（FRA），其核心职责就是确保铁路安全可用。在 FRA 网站上（https://www.fra.dot.gov/）阅读了一些报告和情况简报后，你发现大部分报告都显示，列车事故发生的原因是轨道养护不佳或人为失误。

你对人为失误更感兴趣，所以决定深入挖掘这一点。你发现 FRA 在铁路员工和安全方面发布了大量的报告。你找到一份关于铁路工人睡眠类型的报告（http://catalog.data.gov/dataset/work-schedules-and-sleep-patterns-of-railroad-employees-train-and-engine-service），可以部分解释人为失误发生的原因。你还找到联邦法规关于对铁路员工做药物和酒精测试的一些资料（http://www.fra.dot.gov/eLib/details/L02699）。

现在你可能有更多的问题，你可以将范围缩小，详细阐述你真正想了解的问题。现在你的问题可能会变成“喝酒导致的铁路事故发生频率是多少”或者“火车工程师加班或劳累过

度的频率是多少”。你已经有了初步的可信数据集，还可以在研究过程中致电 FRA 了解更多信息。

6.5.3 足球运动员的薪水

再比如说，你对足球（用脚踢的足球，不是猪皮做的橄榄球）运动员的薪水感兴趣。这些运动员能赚多少钱，每个运动员对球队的影响有多大？

初次搜索之后，你发现数据太杂，决定专注研究某一个联赛。比方说你选择英超联赛。你在一个可能从没听说过的网站上找到了英超俱乐部的薪水列表（http://www.tsmplug.com/football/premier-league-player-salaries-club-by-club/）。看来作者已经编辑好了每一支球队的列表，以及每一名球员的薪水列表（http://www.tsmplug.com/football/man-city-players-salaries-2014/）。为了更好地理解数据的来源，并保证数据源可信，你应该联系页面中给出的作者，以获取更多信息。

如果你同时搜索球员代言，可能会找到这个统计表（http://www.statista.com/statistics/266636/best-paid-soccer-players-in-the-2009-2010-season/），里面列出了顶薪足球运动员的代言费用和薪水数据。你可能也想去联系作者，询问是否有最新的代言费用数据，可以和最新的赛季作对比。

现在你已经有了薪水数据，你还想了解一些统计数据，看看顶薪运动员究竟有多优秀。你在英超联赛网站（http://www.premierleague.com/content/premierleague/en-gb/players/index.html）上找到一些球员统计数据。这可能是你只能用网络抓取来获取的数据（第 11 章会有更多关于网络抓取的内容），但你知道数据来源是可靠的。继续搜索球员统计数据，你可能会在 top assists 网站（http://www.espnfc.com/barclays-premier-league/23/statistics/assists）上找到更多数据。你还可以分析点球统计数据（http://eplreview.com/statistics-penalty.htm）。同样，你应该调查任何数据源的有效性，这一点并不容易验证。

现在你可以开始数据分析，计算每一名球员进球、红牌和点球的价值！

6.5.4 童工

最后，我们来探索一个本书后续章节将要回答的问题。我们将专注研究国际童工危机。当思考国际话题时，我们立刻想到要寻找国际组织。

我们发现 UNICEF 的开放数据网站致力于发布童工报告（http://data.unicef.org/child-protection/child-labour.html）。事实上，UNICEF 拥有全球妇女儿童健康状况的全部数据集（http://mics.unicef.org/）。这些数据集可能对回答类似“早婚对童工率是否有影响？”这样的问题很有帮助。

在寻找政府数据时，我们找到了美国劳工部关于全球童工的年度报告（https://www.dol.gov/agencies/ilab/resources/reports/child-labor）。这些报告可以用来与 UNICEF 的数据集相互对照。

另外，我们还找到国际劳工组织（ILO）关于童工的趋势报告（http://www.ilo.org/ipec/

Informationresources/WCMS_IPEC_PUB_23015/lang--en/index.htm）。ILO 报告似乎给出了许多不同数据集的链接，应该是童工历史数据的很好参考。

我们还汇总了下面几章会用到的几个数据集。我们将这些数据集都放在数据仓库中（https://github.com/jackiekazil/data-wrangling），以便后续使用。

前面已经探讨了如何发现问题并搜索资源，下面我们来看一下数据存储。

6.6 数据存储

找到数据之后，你需要把数据保存下来！有些时候，你得到的数据是干净的、易于访问的、机器可读的格式。其他时候，你可能想用另一种方法来保存数据。当你第一次从 CSV 或 PDF 中提取数据的时候，我们会讲到几种数据存储工具，或者，你可以等数据完全处理并清洗完成后再进行存储（我们会在第 7 章讲到数据清洗的内容）。

我应该把数据保存在哪里？

最开始的问题是，要将数据保存到其他地方，还是留在最开始提取的文件中。这有一系列问题可以帮你回答这个问题。

- 你能否用简单的文档阅读器（例如 Microsoft Word）打开数据集，同时不会造成计算机死机？
- 数据看起来是否具有良好的标签和结构，让你可以方便提取出每一段信息？
- 如果需要不止一台电脑来处理数据的话，数据的保存和移动是否方便？
- 能否利用 API 实时访问数据，这样你就能在线获取需要的数据？

如果所有问题的回答都是“是”，你可能不必担心保存数据的问题。如果你的回答有“是”有“否”的话，可能需要将数据保存在数据库或平面文件（flat file）中。如果所有问题的回答都是“否”，继续读下去，我的朋友，我们为你提供了解决方法！

假设你的数据集各不相同——这里的一个文件，那里的一份报告。其中一些很容易下载和访问，但其他的你可能需要从网络上复制或抓取。第 7 章和第 9 章中会讲到如何清洗与合并数据集，但现在我们来谈一谈如何将数据保存在共享位置。



如果你要用的数据集来自多台电脑，建议你把它们都保存在网络或互联网中（你好，云计算！），或者保存在移动硬盘或 U 盘中。当你和团队合作时，团队成员可能会从不同地点或不同电脑访问数据，一定要记住这一点。如果你在一台计算机上工作，一定要有数据备份策略。电脑丢失最糟糕的一点就是，你花几个月时间获取并清洗的数据也丢失了。

6.7 数据库简介

数据库——初学生爱，爱极生恨。作为开发人员，你可能会在学习和工作中用到各种类型的数据库。本节不会对数据库做全面介绍，但我们希望对数据库基本概念做一个简要介

绍。如果你已经熟练掌握数据库的使用，可以大致浏览一下本节，继续阅读关于其他存储方案以及何时使用数据库的内容。

你用 Siri 查过手机里的电话号码么？你用过谷歌搜索么？你有没有点击过 Twitter 或 Instagram 里的标签？这些操作都涉及对数据库（或一系列数据库，或数据库缓存）的简单查询和响应。你有一个问题［在 YouTube 上新出了哪些关于猫（Maru）的有趣视频？］，你向一个特定的数据库（YouTube 搜索）提问，得到有趣的响应——可以观赏的搜索结果列表。

在接下来的几节里，我们将简要讲述两种主要的数据库类型，强调了各自的利弊，并对比了二者的优点和缺点。对于数据处理来说，你绝对不需要使用数据库，然而，随着数据处理和分析变得更加复杂，数据库知识及其使用将变得更加重要，可以提高你存储和分析数据的能力。

如果你对数据库感兴趣，我们会讲到用 Python 处理数据库的几个技巧，但显然我们没有足够的时间全面讲述这个话题。我们强烈推荐你根据自己的兴趣去搜索更多的资料、视频和教程。

6.7.1　关系型数据库：MySQL和PostgreSQL

对于来源很多、同时还有各种层次关联性的数据，关系型数据库是很好用的。关系型数据库正如其名：如果你的数据连接类似于家谱，那么关系型数据库可能会适合你，比如 MySQL。

关系型数据库通常使用一系列唯一标识符来匹配数据集。在 SQL 里我们一般把这些标识符叫作 ID。这些 ID 可以被其他数据集所用，用来查询和匹配数据连接。在这些连接好的数据库中，我们可以进行 join 操作，在许多不同的数据库中同时访问连接的数据。我们来看一个例子。

我有一个特别厉害的朋友，叫 Meghan。她有一头黑发，在《纽约时报》工作。在工作之余，她喜欢跳舞、烹饪和教人如何编程。如果我有一个关于朋友的数据库，使用 SQL 代表他们的属性，我可能会这样分表：

```
**friend_table: ❶
friend_id ❷
friend_name
friend_date_of_birth
friend_current_location
friend_birthplace
friend_occupation_id

**friend_occupation_table:
friend_occupation_id
friend_occupation_name
friend_occupation_location

**friends_and_hobbies_table:
friend_id
hobby_id
```

```
**hobby_details_table:
hobby_id
hobby_name
hobby_level_of_awesome
```

❶ 在我的朋友数据库中，每一部分（以 ** 表示）都是一个*表*。在关系型数据库中，表通常用来保存特定主题或特定对象的信息。

❷ 表中包含的每一条信息叫作*字段*。在这个例子中，`friend_id` 字段包含 `friend_table` 中每一位朋友的唯一 ID。

我可以向数据库提问：Meghan 的爱好是什么？想要获取这个信息，我需要对数据库说："嘿，我要查询我的朋友 Meghan。她住在纽约，这是她的生日，你能告诉我她的 ID 吗？"对于这条查询，SQL 数据库返回的是她的 `friend_id`。然后我可以向 `friend_and_hobbies_table` 提问（这个表正确匹配了朋友 ID 和爱好 ID），与这个朋友 ID 匹配的爱好是什么，它会返回由三个新的爱好 ID 组成的列表。

由于这些 ID 都是数字，我想进一步了解它们的含义。我向 `hobby_details_table` 提问："你能告诉我关于这些爱好 ID 的更多内容吗？"它回答："当然可以！一个是跳舞，一个是烹饪，一个是教人如何编程。"啊哈！只利用最开始的朋友描述，我就解开了这个谜题。

创建数据库并向其中导入数据可能涉及许多步骤，但如果你的数据库很复杂，有许多不同的关系，那么搞清楚如何连接这些数据并找到你想要的信息，步骤不应该很复杂。在构建关系型数据库时，花点时间研究关系和属性之间的映射，类似我们在朋友数据库所做的那样。都有哪些不同的数据类型，它们之间是如何映射的？

在关系型数据库*架构*中，通过思考数据的使用频率，我们知道要如何匹配数据。你希望向数据库请求的查询易于回答。由于我们可能会用职业来寻找对应的朋友，所以我们将 `occupation_id` 放在 `friend_table` 表中。

还需要注意的是，关系有许多不同的类型。例如，我有许多朋友的爱好都是烹饪。我们把这种情况称为*多对多*关系。如果我们再添加一个叫作 `pets` 的表，就会新增加一种关系类型——*多对一*关系。这是因为有些朋友养了不止一只宠物，但每只宠物只能有一个主人。我可以使用 `friend_id` 查询每一位朋友的所有宠物。

如果你有兴趣深入学习 SQL 和关系型数据库的内容，我们建议在 SQL 上多花点时间。入门 SQL 可以在"Learn SQL The Hard Way"（http://sql.learncodethehardway.org/）和"SQLZOO"（http://sqlzoo.net/）这两个网站上学习。PostgreSQL 和 MySQL 在语法上有一些细微的差别，但它们的基础知识相同，你可以自己选择学习哪一个。

1. MySQL和Python

如果你熟悉 MySQL（或正在学习 MySQL），想要使用 MySQL 数据库，那么用 Python 连接 MySQL 是很容易的。你需要做的只有两步。第一步，你必须安装 MySQL 驱动程序。第二步，你应该用 Python 发送验证信息（用户名、密码、主机名、数据库名称）。这两步在 Stack Overflow 上都可以找到很多优质的回答（http://stackoverflow.com/questions/372885/how-do-i-connect-to-a-mysql-database-in-python）。

2. PostgreSQL和Python

如果你熟悉 PostgreSQL（或正在学习 PostgreSQL），想要使用 PostgreSQL 数据库，那么用 Python 连接 PostgreSQL 也是很容易的。你也只需要做两步：安装驱动程序，用 Python 连接。

Python 的 PostgreSQL 驱动程序有很多（https://wiki.postgresql.org/wiki/Python），但最流行的是 Psycopg（http://initd.org/psycopg/）。在 Psycopg 的安装页面（http://initd.org/psycopg/docs/install.html）中详细介绍了如何在电脑上运行 Psycopg，在 PostgreSQL 网站上也有关于 Python 如何使用 Psycopg 的详细介绍（https://wiki.postgresql.org/wiki/Psycopg2_Tutorial）。

6.7.2　非关系型数据库：NoSQL

比方说，你喜欢使用数据库这个主意，但映射出所有关系会让你抓狂。可能只是因为你目前没有真正理解数据的连接方式，也可能是因为你用的是平面数据（flat data，也就是说，不必良好映射的无关系数据），或者也可能是因为你对学习 SQL 没有更强烈的兴趣。幸运的是，还有一种适合你的数据库。

NoSQL 以及其他非关系型数据库将数据保存成平面格式（flat format），通常是 JSON 格式。我们在第 3 章中说过，JSON 查找信息的方法很简单。回到上一节关于我朋友的数据，如果我只有保存在节点中的数据，通过这些节点可以查询该朋友的更多信息，那我应该怎么做？数据看起来可能是这样的：

```
{
    'name': 'Meghan',
    'occupation': { 'employer': 'NYT',
                    'role': 'design editor',
                  },
    'birthplace': 'Ohio',
    'hobbies': ['cooking', 'dancing', 'teaching'],
}
```

可以看出，我用一个简单列表就可以给出我朋友的所有属性，无需创建表。

你可能想知道，关系型数据的优点是什么？你问不同的人，得到的回答可能会完全不同——在计算机科学领域，在众多开发人员之中，这是一个激烈争论的话题。我们的观点是，当数据结构包含大量的关系网络时，SQL 对快速查询做出了许多改进。非关系型数据库在速度、可用性和复用方面做出了许多改进。

最后，如果你对学习某一种数据库有更强烈的兴趣，可以让兴趣帮你做决定，但不要现在就确定数据库的格式。如果你需要在关系型数据库和非关系型数据库之间迁移，有许多工具可以帮你完成这一任务 [1]。

注 1：对于在 SQL 和 NoSQL 数据库之间的迁移，更多内容可查阅 Matt Asay 关于将 Foursquare 从关系型数据库迁移到 NoSQL 数据库的文章（http://www.techrepublic.com/blog/the-enterprise-cloud/migrating-from-a-relational-to-a-nosql-cloud-database/）。另外，Quora 上还有许多关于反向迁移的文章（https://www.quora.com/How-do-I-migrate-data-from-a-MongoDB-to-MySQL-database-Can-it-be-done-in-a-real-time-scenario-What-are-the-pros-and-cons-for-each-migration-Which-one-do-you-advice-What-is-your-experience-Any-reference-DB-expert-who-can-do-it）。

MongoDB和Python

如果你的数据具有非关系型数据库结构，或者你希望在实践中学习，那么用 Python 连接 NoSQL 数据库是非常简单的。虽然有很多选择，但最流行的 NoSQL 数据库框架之一是 MongoDB（http://mongodb.org/）。要使用 MongoDB，你需要首先安装驱动程序（http://docs.mongodb.org/ecosystem/drivers/python/），然后用 Python 来连接。在 PyCon 2012 上有一个很棒的演讲："Getting Started with MongoDB"（https://github.com/behackett/presentations/tree/master/pycon_2012），你可以从这里开始学习 MongoDB 以及如何用 Python 连接。

6.7.3 用Python创建本地数据库

开始学习数据库和 Python 最简单的方法就是，使用一个简单的库帮你快速上手。对于本书来说，我们推荐学习 Dataset 库（http://dataset.readthedocs.io/）。Dataset 是一个包装库（wrapper library），可以将可读的 Python 代码翻译成要处理的数据库代码，以加快开发速度。

如果你已有一个 SQLite、PostgreSQL 或 MySQL 数据库，可以参考快速入门指南（https://dataset.readthedocs.io/en/latest/quickstart.html）直接连接。如果你还没有上述数据库之一，在使用这个工具时它会为你创建一个。我们来看一下如何让它在你的电脑上运行。

你要做的第一件事是安装 Dataset（http://dataset.readthedocs.io/en/latest/install.html）。如果你已经安装了 pip，那么只需要输入 `pip install dataset`。

然后你需要确定用到的后端。如果你已经在用 PostgreSQL 或 MySQL，只需要用对应的语法创建一个新的数据库。如果你对数据库不太熟悉，那我们就用 SQLite。首先，下载操作系统对应的 SQLite 二进制文件（http://www.sqlite.org/download.html）。打开下载的文件，按照安装说明一步步安装。

打开终端，切换（`cd`）到保存 Python 数据处理脚本的项目文件夹。输入以下代码来创建新的 SQLite 数据库：

```
sqlite3 data_wrangling.db
```

你应该会看到以 `sqlite>` 开头的提示符，提示你输入 SQL 语句。你已经确认电脑上可以运行 sqlite3，可以在终端输入 `.q` 退出 SQLite。退出后，列出当前文件夹的所有文件。现在你应该会看到一个名为 data_wrangling.db 的文件——那就是你的数据库！

安装好了 SQLite，运行了第一个数据库之后，现在要开始使用 Dataset 了。在 Python 中运行以下代码：

```
import dataset

db = dataset.connect('sqlite:///data_wrangling.db')

my_data_source = {
  'url':
  'http://www.tsmplug.com/football/premier-league-player-salaries-club-by-club/',
  'description': 'Premier League Club Salaries',
```

```
  'topic': 'football',
  'verified': False,
} ❶

table = db['data_sources'] ❷
table.insert(my_data_source) ❸

another_data_source = {
  'url':
  'http://www.premierleague.com/content/premierleague/en-gb/players/index.html',
  'description': 'Premier League Stats',
  'topic': 'football',
  'verified': True,
}

table.insert(another_data_source)

sources = db['data_sources'].all()  ❹

print sources
```

❶ 创建一个 Python 字典，里面是我们要保存的数据。我们要保存的是足球研究的数据源。我们添加的信息有主题、描述、URL 以及我们是否对数据做过核实。
❷ 创建名为 data_sources 的新表。
❸ 将第一个数据源插入新表。
❹ 显示我们保存在 data_sources 表中的所有数据源。

你已经利用 SQLite 创建了第一个关系型表，并完成了 Python 与数据库之间的第一次交互。随着本书内容的深入，你将会向数据库中添加更多的数据和表。将所有数据保存到一处，可以让你数据结构清晰，让你的研究更加专注。

6.8 使用简单文件

如果你的数据集很小，很可能简单文件就可以满足要求，不必使用数据库。你可能想浏览一下第 7 章，在保存之前先用数据清洗技术处理一下，但把数据保存成 CSV 文件或其他简单文件格式是完全可以的。我们用来导入 CSV 的 csv 模块（见 3.1.1 节）也有许多好用的写入类（https://docs.python.org/2/library/csv.html#writer-objects）。

在使用简单文件时，你主要考虑的是确保访问和备份文件都比较方便。要满足这些需求，你可以将数据保存在共享网盘或云服务（Dropbox、Box、Amazon、Google Drive）中。这些服务通常都会提供备份选项和管理能力，同时还能够分享文件。在“哎呀，我把数据文件覆盖了”时，这是非常有用的。

6.8.1 云存储和Python

根据你选择的云存储方案，你应该研究一下用 Python 获取数据的最佳方法。Dropbox 对 Python 的支持很好，网站上的“Python 快速入门指南”（https://www.dropbox.com/developers-v1/core/start/python）很不错。Google Drive 要复杂一些，但“Python 快速上手

指南”（https://github.com/googledrive/python-quickstart）可以帮你完成初步的设置。Google Drive 还有一些 Python API 包装器，比如 PyDrive（https://github.com/googledrive/PyDrive），可以让你在不太会用 Python 的情况下使用 Google Drive。要管理 Google Drive 上的电子表格，我们强烈推荐 GSpread（https://github.com/burnash/gspread）。

如果你有自己的云服务器，可能需要研究连接云服务器的最佳方法。Python 有内置的 URL 请求方法、FTP（文件传输协议）方法和 SSH/SCP（Secure Shell/Secure Copy）方法，都包含在 Python 标准库（stdlib）中。在第 14 章中我们还会讲到管理云服务的一些有用库。

6.8.2 本地存储和Python

数据存储最简单也是最直接的方法就是本地存储。用一行 Python 代码就可以打开文件系统中的文档（`open` 命令，https://docs.python.org/2/library/functions.html#open）。在处理数据时，你还可以用内置的 `file.write` 方法修改并保存为新文件。

6.9 其他数据存储方式

数据存储还有许多有趣的新方式，和前面讲过的都不一样。根据你的使用案例，存储数据可能有更好的方式。下面是两种有趣的存储方式。

- *层次型数据格式*（HDF）
  HDF 是基于文件的可扩展数据解决方案，可将大型数据库快速存储至文件系统（本地或其他位置）。如果你已经很熟悉 HDF，Python 有一个 HDF5 驱动程序 h5py（http://www.h5py.org/），可以将 Python 与 HDF5 相连接。

- Hadoop
  Hadoop 是一个大数据分布式存储系统，可以跨集群存储并处理数据。如果你已经用过 Hadoop，或者熟悉 Hadoop，在 Cloudera 网站上有一篇“Hadoop 上 Python 框架指南”（http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/），还有一些容易上手的代码示例。

6.10 小结

恭喜！你已经搞定了项目面临的几个最大问题：我怎么能找到有用的数据？我怎么访问并保存数据？我们希望你对获取的数据源有信心，并相信你第一个数据集的真实性。我们也希望你对数据备份和数据存储有一个可靠的计划。

你可以将本章学习的技术应用到以后的数据集上，即使是在数据网站上花几个小时研究脑海中突然出现的问题。

现在你应该有信心做好以下事情：

- 判断你找到数据集的价值和用途
- 拿起电话寻求更多信息
- 要回答一个问题，知道首先去哪里寻找数据

- 轻松实现安全存储数据的方法
- 核实你找到的数据
- 构建数据的关系模型

你还第一次接触到表 6-2 中的这些概念。

表6-2：Python编程新概念和新库

概念/库	作用
关系型数据库（例如 MySQL 和 PostgreSQL）	轻松存储关系型数据
非关系型数据库（例如 MongoDB）	以平面方式存储数据
SQLite（https://www.sqlite.org/）安装和使用	基于 SQL 的易用存储，适用于简单项目
Dataset（https://dataset.readthedocs.org/en/latest/）安装和使用	易用的 Python 数据库包装器

在后续章节中，你还会更多地用到所有这些技术。在下一章里，你将学习清洗数据，利用代码发现异常，编写完整的脚本或程序，这样你就可以分析数据，并输出结果与全世界分享。

第 7 章

数据清洗：研究、匹配与格式化

数据清洗并不是最迷人的工作，却是数据处理的重要组成部分。要想成为数据清洗专家，需要严谨的态度，以及对所研究领域全面系统的知识。学会如何正确地清洗数据并汇总，可以让你在研究领域中脱颖而出。

Python 的设计很适合数据清洗，它可以创建函数处理相同的规律，减少重复性工作。根据我们目前所学的代码知识，学会用脚本和代码处理重复性的问题，可以节省数小时的体力劳动，只需要运行一次脚本就可以完成。

本章我们将学习如何用 Python 清洗数据和格式化数据。我们还会用 Python 寻找数据集中的重复数据和错误。在下一章里我们会继续学习数据清洗，特别是清洗过程自动化和清洗后的数据存储。

7.1　为什么要清洗数据

对于你获取的数据，有些可能格式良好，方便使用。如果真是这样的话，那你很幸运！大部分数据即使清洗过，也会有格式不一致和可读性的问题，例如首字母缩写或描述性标题不匹配，特别是数据来自多个数据集。除非你在数据格式化和标准化上花点工夫，否则数据不可能正确合并，也就没有用处了。

> 清洗数据可以让数据更容易存储、搜索和复用。我们在第 6 章中学过，先清洗数据，再把数据保存到适当的模型中会容易得多。想象一个数据集中有很多列（或字段），应该保存成特定的数据类型，比如日期、号码或电子邮件地址。如果你能将预期格式标准化，清洗或删除不合格的数据，就可以保证数据的一致性，在以后需要查询数据集时也不用做大量工作。

如果你想展示你的发现并发布数据，就要发布清洗过的版本。这样其他数据处理人员就能轻松导入并分析数据。你还可以在发布最终数据集的同时发布原始数据，并说明你是如何一步步清洗数据并将其归一化的。

在清洗数据的过程中，我们希望记下清洗过程的每一步，这样就可以在研究中为我们的数据集及其使用方法申辩，同时也可以方便我们自己以及其他人的后续使用。通过记录清洗过程，在遇到新的数据时我们可以重复整个过程。

> 如果你用 IPython 与数据交互，一个强大的工具是 IPython 的魔法命令，比如 `%logstart`（https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-logstart）可用于记录日志，`%save`（https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-save）可以保存当前会话供以后使用。这样你就可以在 Python 终端里创建脚本，而不仅仅是一行行的代码。随着对 Python 的进一步学习，你可以完善脚本并与其他人分享。想了解更多 IPython 的内容，可查阅附录 F。

下面我们开始学习数据清洗的基础知识，学习如何格式化数据，以及如何将多个数据集正确地匹配在一起。

7.2　数据清洗基础知识

如果已经完成了前面几章的代码练习，你就已经用到了一些数据清洗的概念。在第 4 章，我们从 Excel 工作表里导出数据，并创建一个字典来表示这些数据。修改数据使其满足新的标准化数据格式，这个过程就是数据清洗。

我们已经研究过 UNICEF 发布的与童工有关的数据集（见 6.5.4 节），下面我们深入研究 UNICEF 的原始数据。大部分 UNICEF 报告的原始数据集都来自多指标类集调查（Multiple Indicator Cluster Surveys，MICS，http://mics.unicef.org/surveys）。这些调查是由 UNICEF 工作人员和志愿者做的家庭层面调查，用于对世界各地妇女和儿童生活状况的研究。浏览最新的几份调查，我们提取出津巴布韦最新的 MICS 数据进行分析。

在开始分析之前，我们首先需要以教育和研究为目的请求访问 UNICEF 网站，然后下载最新的调查。获取访问权限之后（大约需要一天的时间），我们就可以下载原始数据集。大多数 MICS 原始数据是 SPSS 格式或 .sav 文件。SPSS 是社会科学家用来保存并分析数据的程序。对于某些社会科学统计数据来说，它是很好用的工具，但并不太适合用 Python 来处理。

为了将 SPSS 文件转换成可用的格式，我们首先使用开源项目 PSPP（https://www.gnu.org/software/pspp/）来查看数据，然后用几个简单的 R 命令将 SPSS 数据转换成 .csv 文件（http://bethmcmillan.com/blog/?p=1073），这样 Python 处理起来会比较方便。还有许多优秀的项目，可以用 Python 与 SPSS 文件交互（https://pypi.python.org/pypi/savReaderWriter），但其安装和操作都要比 R 命令复杂。你可以在本书仓库（https://github.com/jackiekazil/data-wrangling）中找到生成的 CSV 文件。

仔细观察文件及其包含的数据，我们从这里开始数据清洗过程。数据清洗的第一步通常是简单的目视分析。我们仔细观察文件，看看能发现什么！

7.2.1 找出需要清洗的数据

数据清洗的第一步是，观察数据字段，仔细寻找不一致的地方。如果在数据清洗之初就可以使数据看起来更加干净的话，你会更容易找到在数据归一化过程中需要解决的最初问题。

我们来看一下 mn.csv 文件。文件中包含原始数据，并用首字母缩写作为标题，这些缩写的含义可能很好翻译。我们来看一下 mn.csv 文件的列标题：

```
"","HH1","HH2","LN","MWM1","MWM2", ...
```

每一项都代表调查中的一个问题或数据，我们想要的是可读性更强的版本。通过谷歌搜索，我们在分享 MICS 数据的世界银行网站（http://microdata.worldbank.org/index.php/catalog/1794/datafile/F5）上找到了这些标题的具体含义。

> 花点时间研究世界银行网站上有没有缩写一览表，可以帮你更好地完成数据清洗工作。你还可以给该机构打电话，询问是否有易于使用的缩写列表。

利用在第 11 章即将学到的一些网络抓取技术，我们可以得到一个 CSV 文件，里面包含这些标题及其英文释义，以及世界银行在采集这些 MICS 数据时所问的问题。我们已将网络抓取生成的新标题放在本书仓库中（mn-headers.csv）。我们希望将这些标题与调查数据一一对应，这样就有了可读性较强的问题和答案。我们有几种方法可以做到这一点。

1. 替换标题

想要提高标题可读性，最明显而直接的方法就是将短标题替换成易于理解的长标题。如何用 Python 做标题替换呢？首先，需要用第 3 章学过的 `csv` 模块导入 mn.csv 和 mn-headers.csv 两个文件（参见下面的导入代码）。在本章和下一章中，你可以在脚本中写代码，也可以在终端里（比如 IPython）写代码。这样你可以先与数据交互，然后再将其保存到文件中：

```
from csv import DictReader

data_rdr = DictReader(open('data/unicef/mn.csv', 'rb'))
header_rdr = DictReader(open('data/unicef/mn_headers.csv', 'rb'))

data_rows = [d for d in data_rdr] ❶
header_rows = [h for h in header_rdr]

print data_rows[:5] ❷
print header_rows[:5]
```

❶ 本行代码将可迭代对象 `DictReader` 写入一个新列表，这样我们可以保存数据并重复使用。我们用到了列表生成式，只需要一行简单、清晰、易读的代码即可实现。

❷ 本行代码打印出数据的一个切片，用的是 Python 列表的 `slice` 方法，显示新列表的前 5 个元素，对列表内容有一个初步了解。

在第 4 行代码中，我们用到了 Python 的列表生成式（list comprehension）函数。Python 列表生成式的格式如下：

```
[func(x) for x in iter_x]
```

列表生成式的首尾是列表的方括号。它用到一个可迭代对象（*iter_x*），将 *iter_x* 的每一行或每一个值传入 *func(x)*，用返回值创建一个新列表。这里我们没有用到列表生成式的函数，只用到了每一行的当前值。在后续章节中，我们会将可迭代对象的每一行或每一个值传入一个函数，对数据进行清洗或修改，然后再添加到新列表中。列表生成式是易读易用的语法一个很好的例子，这也是 Python 广为人知的特点。用 `for` 循环也可以实现相同的功能，但是代码量更大：

```
new_list = []
for x in iter_x:
    new_list.append(func(x))
```

可以看出，列表生成式的代码比较简短，性能更好，也更节省内存。

我们希望将 `data_rows` 中字典标题替换为文件里可读性较强的标题。从输出里可以看出，`header_rows` 字典里同时包含短标题和长标题。短标题包含在 `Name` 字段，可读性更强的长标题包含在 `Label` 字段。利用一些 Python 字符串方法可以将二者轻松匹配在一起：

```
for data_dict in data_rows: ❶
    for dkey, dval in data_dict.items(): ❷
        for header_dict in header_rows: ❸
            for hkey, hval in header_dict.items():
                if dkey == hval: ❹
                    print 'match!'
```

❶ 遍历每一条数据记录。我们将用每一个字典的键与标题匹配。
❷ 遍历每一行数据的键和值，这样就可以将所有键替换成可读性更强的标题标签（要查看数据字典里每一个键值对，我们用的是 Python 字典的 `items` 方法）。
❸ 遍历所有标题行，这样我们可以得到可读性更强的标签。这并不是最快的方法，但可以保证不会有遗漏。
❹ 如果数据列表的键（`MWB3`、`MWB7`、`MWB4`、`MWB5`…）和标题字典的值相同，那么打印 `'match!'` 表示二者相互匹配。

运行代码，我们发现有很多匹配项。下面我们来看一下，能否用类似的逻辑将标题替换成更好的标题。我们知道将二者匹配是相当容易的。但我们只找到了想要匹配的那些行。我们来看一下能否找到一种方法，将 `data_rows` 每一项的键与 `header_rows` 每一项的值相匹配：

```
new_rows = [] ❶

for data_dict in data_rows:
    new_row = {} ❷
    for dkey, dval in data_dict.items():
        for header_dict in header_rows:
            if dkey in header_dict.values(): ❸
                new_row[header_dict.get('Label')] = dval ❹
    new_rows.append(new_row) ❺
```

❶ 创建一个新列表，里面包含的是清洗过的行数据。
❷ 为每一行创建一个新字典。
❸ 这里我们用的是字典的 `values` 方法，而不是遍历标题行的所有键值对。这一方法返回的是仅由字典的值组成的列表。我们还用到了 Python 的 `in` 方法，用来测试一个对象是否包含在某个列表中。在本行代码中，对象是我们的键，或缩写字符串，而列表是标题字典的值组成的列表（里面包含缩写短标题）。如果本行代码为真，我们就找到了一个匹配行。
❹ 每找到一个匹配，都将其添加到 `new_row` 字典中。将字典的键设置为标题行中 `Label` 对应的值，也就是将短标题（`Name` 对应的值）替换为可读性更好的长标题（`Label` 对应的值），并将字典的值设置为数据行的值（`dval`）。
❺ 将清洗过的新数据添加到新列表中。这样做是为了保证我们找到了所有的匹配，然后再运行后面的代码。

将新列表第一项打印出来，可以看出，我们已经成功提高了数据的可读性：

```
In [8]: new_rows[0]
Out[8]: {
    'AIDS virus from mother to child during delivery': 'Yes',
    'AIDS virus from mother to child during pregnancy': 'DK',
    'AIDS virus from mother to child through breastfeeding': 'DK',
    'Age': '25-29',
    'Age at first marriage/union': '29',...
```

要检查函数的缩进是否正确，一个简单的方法是观察具有相同缩进的其他代码。不停问自己：其他代码这一步的逻辑是什么？我的代码应该何时继续下一步？

对于数据清洗问题，解决方法远不止一种。那么对于我们标题可读性较差的问题，我们来看一下能否用其他方法解决。

2. 合并问题与答案

修复标签问题的另一种方法是 Python 的 `zip` 方法：

```
from csv import reader ❶

data_rdr = reader(open('data/unicef/mn.csv', 'rb'))
header_rdr = reader(open('data/unicef/mn_headers.csv', 'rb'))

data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr]

print len(data_rows[0]) ❷
print len(header_rows)
```

❶ 这次我们用的是简单的 `reader` 类，而不是 `DictReader`。`reader` 为每一行创建的是一个列表，而不是一个字典。由于我们要用的是 `zip` 方法，需要的是列表而不是字典，这样我们就可以将标题列表与数据列表合并在一起。

❷ 这几行代码创建标题列表和数据列表，并查看它们的长度是否相同。

啊呀——打印结果显示，数据列表长度与标题列表长度并不相同！我们的数据有 159 行，而标题列表中有 210 个标题。这说明，与津巴布韦数据集相比，MICS 在其他国家可能问了更多问题，或者提供了更多的备选问题。

我们需要进一步研究数据集中都用了哪些标题，哪些标题是不需要的。我们仔细观察一下，找出没有正确对应的那些标题：

```
In [22]: data_rows[0]
Out[22]: ['',
 'HH1',
 'HH2',
 'LN',
 'MWM1',
 'MWM2',
 'MWM4',
 'MWM5',
 'MWM6D',
 'MWM6M',
 'MWM6Y',
... ]

In [23]: header_rows[:2]
Out[23]: [
 ['Name', 'Label', 'Question'],
 ['HH1', 'Cluster number', '']]
```

好吧，我们可以很清楚地看到，data_rows 的第二行与 header_rows 中索引数为 1 的值对应。找出不匹配的那些行，然后将它们从 header_rows 中删除，这样我们就可以将数据正确地合并：

```
bad_rows = []

for h in header_rows:
    if h[0] not in data_rows[0]: ❶
        bad_rows.append(h) ❷

for h in bad_rows:
    header_rows.remove(h) ❸

print len(header_rows)
```

❶ 测试标题行的第一个元素（标题的缩写版本）是否包含在数据列表的第一行中（所有的标题缩写）。

❷ 将不匹配的标题行添加到新列表 bad_rows 中。下一步我们将利用这个列表来判断需要删除哪些行。

❸ 利用列表的 remove 方法从列表中删除指定的一行。当你能够指出想从列表中删除的某一行（或某些行）时，这个方法往往是很有用的。

啊哈！现在我们已经几乎完成匹配了。我们的数据列表中有 159 个值，标题列表中有 150 个值。下面我们来看一下，为什么标题列表里不需要这 9 个匹配的标题：

```
all_short_headers = [h[0] for h in header_rows] ❶

for header in data_rows[0]: ❷
    if header not in all_short_headers: ❸
        print 'mismatch!', header ❹
```

❶ 利用 Python 列表生成式采集每一个标题行的第一个元素，生成一个由所有标题缩写组成的列表。

❷ 遍历数据集中的所有标题，检查哪些没有包含在清洗后的标题列表中。

❸ 挑出不包含在缩写列表中的标题。

❹ 用 `print` 语句打印出所有的不匹配项。如果你需要将两个字符串打印在同一行中，只需要在中间加一个 `,`，就可以用空格将两个字符串连接在一起。

运行代码，输出应该是这样的：

```
mismatch!
mismatch! MDV1F
mismatch! MTA8E
mismatch! mwelevel
mismatch! mnweight
mismatch! wscoreu
mismatch! windex5u
mismatch! wscorer
mismatch! windex5r
```

从我们目前所知和输出结果来看，只有几个不匹配标题（带有大写字母的那几个）是需要处理的。小写的那些标题是用于 UNICEF 内部方法的，与我们要研究的问题无关。

由于我们创建的从世界银行网站采集标题的网络爬虫没有找到 `MDV1F` 和 `MTA8E` 这两个变量，所以我们需要用 SPSS 阅读器来查看它们的含义。（另一种方法是删掉这几行，继续下一步。）

> 在处理原始数据时，有时你会发现，想要将数据转换成可用的格式，你需要舍弃不需要的数据或难以清洗的数据。归根结底，决定因素并不在于是否懒惰，而在于数据对你的问题是否重要。

打开 SPSS 阅读器后可以看到，`MDV1F` 对应的标签是“如果妻子不忠，殴打妻子是否合理？”，同时还与关于家庭暴力的其他一系列长问题相对应。其他问题也有一些是关于家庭暴力的，所以最好保留这个问题。研究 `MTA8E` 这个标题发现，它对应的一系列不同的问题，都是关于某人吸食烟草的类型。我们已经将两个标题都添加到新的文件（mn_headers_updated.csv）中。

现在重新运行之前的代码，这次用的是修改后的标题文件。

我们来看一下所有的代码，修改其中几处，这样就可以将标题和数据合并在一起。下面这个脚本需要大量内存，如果你计算机的 RAM 小于 4GB 的话，我们建议在 IPython 终端或 IPython notebook 中运行，这样可以避免出现段错误（segmentation fault）：

```
from csv import reader

data_rdr = reader(open('data/unicef/mn.csv', 'rb'))
header_rdr = reader(open('data/unicef/mn_headers_updated.csv', 'rb'))

data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr if h[0] in data_rows[0]] ❶

print len(header_rows)

all_short_headers = [h[0] for h in header_rows]

skip_index = [] ❷

for header in data_rows[0]:
    if header not in all_short_headers:
        index = data_rows[0].index(header) ❸
        skip_index.append(index)

new_data = []

for row in data_rows[1:]: ❹
    new_row = []
    for i, d in enumerate(row): ❺
        if i not in skip_index: ❻
            new_row.append(d)
    new_data.append(new_row) ❼

zipped_data = []

for drow in new_data:
    zipped_data.append(zip(header_rows, drow)) ❽
```

❶ 使用列表生成式快速删除不匹配的标题。可以看到，我们还在列表生成式中使用了 `if` 语句。本行代码的作用是，如果标题行第一个元素（标题缩写）包含在数据行的标题中，那么将该标题行添加到新列表中。

❷ 创建新列表，用来保存我们不希望保存的数据行的索引编号。

❸ 利用 Python 列表的 `index` 方法，返回我们不需要的索引编号，因为这些索引编号对应的标题没有包含在缩写列表中。下一行代码会将不匹配标题行的索引编号保存下来，这样我们就可以不采集这些数据。

❹ 对保存调查数据的列表做切片，只选取其中的数据行（除了第一行外的所有行），然后对其进行遍历。

❺ 利用 `enumerate` 函数找出不需要保存的那些数据行。这个函数接受一个可迭代对象（本例中是数据行列表），返回每一个元素的索引编号和值。该函数将返回的第一个值（索引编号）赋值给 `i`，将数据值赋值给 `d`。

❻ 检查并确保索引编号不在我们不希望保存的列表中。

❼ 检查完数据行中的每一项（或每一“列”）之后，将新的数据条目添加到 `new_data` 列表中。

❽ 将完全匹配的标题和数据合并在一起，并添加到新数组 `zipped_data` 中。

现在我们可以将新数据集的一行打印出来，看看与我们的预期是否相同：

```
In [40]: zipped_data[0]
Out[40]: [(['HH1', 'Cluster number', ''], '1'),
(['HH2', 'Household number', ''], '17'),
(['LN', 'Line number', ''], '1'),
(['MWM1', 'Cluster number', ''], '1'),
(['MWM2', 'Household number', ''], '17'),
(['MWM4', "Man's line number", ''], '1'),
(['MWM5', 'Interviewer number', ''], '14'),
(['MWM6D', 'Day of interview', ''], '7'),
(['MWM6M', 'Month of interview', ''], '4'),
(['MWM6Y', 'Year of interview', ''], '2014'),
(['MWM7', "Result of man's interview", ''], 'Completed'),
(['MWM8', 'Field editor', ''], '2'),
(['MWM9', 'Data entry clerk', ''], '20'),
(['MWM10H', 'Start of interview - Hour', ''], '17'),
....
```

我们已经将所有的问题和回答保存在元组中，每一行中所有的标题和数据都是匹配好的。为了检查这些匹配是否正确，我们来看一下该行的结尾：

```
(['TN11', 'Persons slept under mosquito net last night',
'Did anyone sleep under this mosquito net last night?'], 'NA'),
(['TN12_1', 'Person 1 who slept under net',
'Who slept under this mosquito net last night?'], 'Currently married/in union'),
(['TN12_2', 'Person 2 who slept under net',
'Who slept under this mosquito net last night?'], '0'),
```

数据看起来很奇怪。似乎出现了一些匹配错误。我们来做一个真实性核查（reality check），利用刚学过的 `zip` 方法来检查标题是否正确匹配：

```
data_headers = []

for i, header in enumerate(data_rows[0]): ❶
    if i not in skip_index: ❷
        data_headers.append(header)

header_match = zip(data_headers, all_short_headers)   ❸

print header_match
```

❶ 遍历数据列表中的所有标题。
❷ 利用 `if...not in...` 语句，只有当索引编号不包含在 `skip_index` 中时才会返回 `True`。
❸ 将两个标题列表合并在一起，这样我们可以观察寻找不匹配项。

啊哈！你发现错误了吗？

```
....
('MHA26', 'MHA26'),
('MHA27', 'MHA27'),
('MMC1', 'MTA1'),
('MMC2', 'MTA2'),
....
```

在这一项之前的匹配都是正确的，在这一项之后，我们的标题文件和数据文件的问题顺序出现了不一致。由于 zip 方法不会改变列表元素的顺序，我们必须首先调整标题的顺序，使其与数据集的顺序一致。下面是修改后的代码，尝试将数据正确匹配：

```
from csv import reader

data_rdr = reader(open('data/unicef/mn.csv', 'rb'))
header_rdr = reader(open('data/unicef/mn_headers_updated.csv', 'rb'))

data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr if h[0] in data_rows[0]]

all_short_headers = [h[0] for h in header_rows]

skip_index = []
final_header_rows = [] ❶

for header in data_rows[0]:
    if header not in all_short_headers:
        index = data_rows[0].index(header)
        skip_index.append(index)
    else: ❷
        for head in header_rows: ❸
            if head[0] == header: ❹
                final_header_rows.append(head)
                break ❺

new_data = []

for row in data_rows[1:]:
    new_row = []
    for i, d in enumerate(row):
        if i not in skip_index:
            new_row.append(d)
    new_data.append(new_row)

zipped_data = []

for drow in new_data:
    zipped_data.append(zip(final_header_rows, drow)) ❻
```

❶ 创建新列表，包含顺序正确的最终标题行。
❷ 利用 else 语句，只将匹配的列添加到列表中。
❸ 遍历 header_rows，直到找到匹配为止。
❹ 检查标题缩写是否匹配。我们用 == 来检查匹配。
❺ 找到匹配后，利用 break 退出 for head in header_rows 循环。这样速度更快，而且不会对结果造成影响。
❻ 将新的 final_header_rows 列表与标题行按顺序正确地合并在一起。

运行新代码，我们看一下第一个数据条目的结尾：

```
([ 'TN12_3', 'Person 3 who slept under net',
'Who slept under this mosquito net last night?'], 'NA'),
(['TN12_4', 'Person 4 who slept under net',
'Who slept under this mosquito net last night?'], 'NA'),
(['HH6', 'Area', ''], 'Urban'),
(['HH7', 'Region', ''], 'Bulawayo'),
(['MWDOI', 'Date of interview women (CMC)', ''], '1372'),
(['MWDOB', 'Date of birth of woman (CMC)', ''], '1013'),
(['MWAGE', 'Age', ''], '25-29'),
```

看起来匹配得很好。我们可以写出更加清晰的代码，但我们已经找到一个好方法，可以保存绝大部分数据并将数据与标题合并在一起，而且速度还很快。

关于你需要的数据完整性，以及在你的项目中需要为数据清洗花费多少精力，你总是需要作出评估。如果你只用其中一部分数据，可能就不需要保存所有数据。如果数据集是你的主要研究来源，那么值得你花费时间和精力来保证数据完整性。

本节我们学习了一些新的工具和方法，可以发现错误或需要清洗的数据，并结合我们学过的 Python 知识和解决问题的方法来解决这些问题。数据清洗第一步工作（替换标题文本）舍弃了一些数据列，保存了其余的数据列，并没有显示标题有缺失。但只要最后得到的数据集中包含我们需要的数据列，这种方法就能满足我们的要求，速度更快，代码量也更少。

在数据清洗过程中思考这几类问题。保持数据完整性是不是很重要？如果是的话，值得花几个小时？有没有简单的方法，既可以适当清洗数据，又可以保存你需要的所有内容？有没有可重复的方法？在清洗数据集时这些问题可以为你提供指导。

现在我们已经有了一个良好的数据列表，我们可以继续学习其他类型的数据清洗。

7.2.2　数据格式化

数据清洗最常见的形式之一，就是将可读性很差或根本无法读取的数据和数据类型转换成可读性较强的格式。特别是如果需要用数据或可下载文件来撰写报告，你需要将其由机器可读转换成人类可读。如果你的数据需要与 API 一起使用，你可能需要特殊格式的数据类型。

对于格式化字符串和数字，Python 为我们提供了大量方法。在第 5 章讲到调试并显示结果时，我们用过 `%r`，作用是以字符串格式或 Unicode 格式给出对象的 Python 表示。Python 还有字符串格式化方法 `%s` 和 `%d`，分别代表字符串和数字。它们通常与 `print` 命令一起使用。

将对象转换成字符串或 Python 表示还有一个更高级的方法，就是利用 `format` 方法。正如 Python 官方文档（https://docs.python.org/2/library/stdtypes.html#str.format）所述，这个方法可以定义一个字符串，并把数据作为参数或关键字参数传入字符串。我们来仔细看一下 `format` 方法：

```
for x in zipped_data[0]:
    print 'Question: {}\nAnswer: {}'.format( ❶
        x[0], x[1]) ❷
```

❶ format 用 {} 表示数据传入的位置，用 \n 换行符来表示换行。
❷ 这里我们传入的是问题和答案组成的元组的前两个值。

你应该会看到像这样的输出：

```
Question: ['MMT9', 'Ever used Internet', 'Have you ever used the Internet?']
Answer: Yes
Question: ['MMT10', 'Internet usage in the last 12 months',
'In the last 12 months, have you used the Internet?']
Answer: Yes
```

很难读懂这是什么意思。我们试着对其进一步清洗。从输出中可以看出，在问题列表中，索引编号为 0 的是缩写，索引编号为 1 的是问题描述。我们希望只用列表的第二部分，可以作为一个很好的标题。我们再试一次：

```
for x in zipped_data[0]:
    print 'Question: {[1]}\nAnswer: {}'.format(
        x[0], x[1])    ❶
```

❶ 这次我们用格式语法 1 来挑出对应索引编号的数据，使输出结果可读性更强。

我们再来看一下输出结果：

```
Question: Frequency of reading newspaper or magazine
Answer: Almost every day
Question: Frequency of listening to the radio
Answer: At least once a week
Question: Frequency of watching TV
Answer: Less than once a week
```

现在可读性就很好了。好耶！下面我们看一下 format 方法的其他用法。当前的数据集中没有很多数字，所以我们用几个示例数字，展示不同数字类型的更多格式化方法：

```
example_dict = {
    'float_number': 1324.321325493,
    'very_large_integer': 43890923148390284,
    'percentage': .324,
}

string_to_print = "float: {float_number:.4f}\n" ❶
string_to_print += "integer: {very_large_integer:,}\n" ❷
string_to_print += "percentage: {percentage:.2%}" ❸

print string_to_print.format(**example_dict) ❹
```

❶ 这里用到了字典，利用键访问字典的值。我们用 : 来分隔键名和格式。.4f 的意思是，将数字转换成浮点数（f），保留四位小数（.4）。
❷ 数字格式不变（键名和冒号），插入逗号（,）作为千位分隔符。
❸ 数字格式不变（键名和冒号），插入百分号（%），小数部分保留两位有效数字（.2）。

❹ 对长字符串调用 `format` 方法，并传入数据字典，用 `**` 将字典拆包（unpack）。将 Python 字典拆包，也就是将字典的键 / 值拆开，拆包后的键和值被传递给 `format` 方法。

> 阅读 Python 格式化的文档和实例（https://docs.python.org/2/library/string.html#format-string-syntax）可以了解更多高级格式化的内容，例如利用 `format` 方法删除不必要的空格、按长度对齐数据和解数学方程。

除了字符串和数字，用 Python 格式化日期也很容易。Python 的 `datetime` 模块中有许多方法，可以格式化 Python 中已有（或生成）的日期，也可以读取任意日期格式，然后创建 Python 对象（`date` 对象、`datetime` 对象、`time` 对象）。

> Python 中日期格式化最常用的方法是 `strftime` 和 `strptime`，将字符串转换成日期最常用的也是这两个方法，如果你用其他编程语言做过日期格式化的话，可能会熟悉这两种格式化方法。想了解更多内容，请阅读关于"`strftime` 和 `strptime` 特性"（https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior）的官方文档。

`datetime` 模块的 `strptime` 方法可以将字符串或数字转换成 Python 的 `datetime` 对象。如果你希望将日期和时间保存到数据库中，或者需要调整时区或增加一个小时，这个方法都很有用。转换成 Python 对象之后，你可以充分利用 Python 处理日期的功能，很容易将其重新转换成人类可读或机器可读的字符串。

我们来看一下 `zipped_data` 列表中保存的采访起止时间数据。首先我们来回顾一下，打印出前几条数据记录，熟悉一下我们即将使用的数据：

```
for x in enumerate(zipped_data[0][:20]): ❶
    print x

.....
(7, (['MWM6D', 'Day of interview', ''], '7'))
(8, (['MWM6M', 'Month of interview', ''], '4'))
(9, (['MWM6Y', 'Year of interview', ''], '2014'))
(10, (['MWM7', "Result of man's interview", ''], 'Completed'))
(11, (['MWM8', 'Field editor', ''], '2'))
(12, (['MWM9', 'Data entry clerk', ''], '20'))
(13, (['MWM10H', 'Start of interview - Hour', ''], '17'))
(14, (['MWM10M', 'Start of interview - Minutes', ''], '59'))
(15, (['MWM11H', 'End of interview - Hour', ''], '18'))
(16, (['MWM11M', 'End of interview - Minutes', ''], '7'))
```

❶ 利用 Python 的 `enumerate` 函数来查看我们需要处理哪些行的数据。

有了全部数据，现在我们需要找到采访开始和结束的具体时间。利用类似的数据，我们可以判断采访时间更可能出现在早上还是晚上，以及采访时长是否会影响回答的数量。我们还可以找出哪个是第一次采访，哪个是最后一次采访，然后计算平均每次采访所用的时间。

我们尝试用 `strptime` 将数据导入 Python 的 `datetime` 对象：

```
from datetime import datetime

start_string = '{}/{}/{} {}:{}'.format( ❶
    zipped_data[0][8][1], zipped_data[0][7][1], zipped_data[0][9][1], ❷
    zipped_data[0][13][1], zipped_data[0][14][1])

print start_string

start_time = datetime.strptime(start_string, '%m/%d/%Y %H:%M') ❸

print start_time
```

❶ 创建一个字符串模板（base string），用于解析多个数据条目中的数据。本行代码使用的是美国人常用的日期格式：月、日、年，然后是小时和分钟。

❷ 数据读取格式如下：`zipped_data[ 第一个数据条目 ][ 数据行编号（由 enumerate 得到）][ 数据本身 ]`。利用第一个数据条目测试，索引号为 8 的那一行是月，索引号为 7 的那一行是日，索引号为 9 的那一行是年。每个元组的第二个元素（`[1]`）是我们需要的数据。

❸ 调用 `strptime` 方法，输入的是一个日期字符串和一个格式字符串，格式字符串的语法可参见 Python 官方文档（https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior）。`%m/%d/%Y` 代表月、日、年，`%H:%M` 代表小时和分钟。这个方法返回一个 Python 的 `datetime` 对象。

> 如果你用 IPython 运行代码，你不需要用 `print` 来查看你感兴趣的每一行内容。常见的做法是，只需输入变量名，就可以在交互式终端中查看输出的内容。你甚至还可以用 Tab 键自动补全。

有了上面的代码，我们可以创建一个通用的日期字符串，然后利用 `datetime` 库的 `strptime` 方法来解析。在我们的数据集里，由于日期数据的每一项都是单独的元素，我们还可以自己创建 Python 的 `datetime` 对象，不必使用 `strptime` 方法。我们来看一下：

```
from datetime import datetime

end_time = datetime( ❶
    int(zipped_data[0][9][1]), int(zipped_data[0][8][1]), ❷
    int(zipped_data[0][7][1]), int(zipped_data[0][15][1]),
    int(zipped_data[0][16][1]))

print end_time
```

❶ 将整数直接传递给 `datetime` 模块的 `datetime` 类，生成一个日期对象。我们传入整数作为参数，整数之间用逗号隔开。

❷ 由于 `datetime` 只接受整数，本行代码将所有的数据转换成整数。`datetime` 输入参数的顺序是年、月、日、时、分，所以我们必须相应调整数据的顺序。

如你所见，利用更少的代码（见上面的例子），我们可以得到采访结束时间的 `datetime` 对象。现在我们有了两个 `datetime` 对象，可以对它们做一些数学运算了！

```
duration = end_time - start_time ❶
```

```
print duration ❷

print duration.days ❸

print duration.total_seconds() ❹

minutes = duration.total_seconds() / 60.0 ❺

print minutes
```

❶ 用结束时间减去开始时间，计算出采访时长。
❷ 打印一个新的 Python 日期类型。这是一个 timedelta 对象。timedelta 会给出两个时间对象的时间差，还可用于改变时间对象，详见 datetime 文档（https://docs.python.org/2/library/datetime.html?highlight=timedelta#datetime.timedelta）。
❸ 利用内置的 days 属性来查看 timedelta 对象里包含了多少天。
❹ 调用 timedelta 对象的 total_seconds 方法，计算时间差包含多少秒。结果精确到微秒。
❺ 计算分钟数，因为 timedelta 对象没有分钟属性。

运行代码，我们知道第一次采访时长 8 分钟——但这是不是采访的平均时长呢？利用刚学的 datetime 模块中的方法解析整个数据集，我们可以计算出采访平均时长。我们前面做了一些简单的 datetime 数学运算，并学习了如何利用数据集创建 Python 的 datetime 对象。下面我们来看一下，能否将这些新的 datetime 对象重新转换成格式化字符串，用在报告中可以提高可读性：

```
print end_time.strftime('%m/%d/%Y %H:%M:%S') ❶

print start_time.ctime() ❷

print start_time.strftime('%Y-%m-%dT%H:%M:%S') ❸
```

❶ strftime 只能输入一个参数，就是你希望显示的日期格式。本行代码输出美国标准时间格式。
❷ Python 的 datetime 对象有一个 ctime 方法，可以根据 C 语言的 ctime 标准输出 datetime 对象。
❸ Python 的 datetime 对象可以用你能想到的任意格式输出字符串。本行代码用的是 PHP 语言常用的时间格式。如果你需要用特殊字符串格式与 API 交互，datetime 模块可以帮到你。

Python 的 datetime 对象非常有用，处理、导入和导出（通过格式化的方法）这种对象都非常简单。根据数据集的不同，你可以利用这些新方法将所有的字符串或 Excel 数据导入并转换成 datetime 对象，做统计分析或取平均值，然后再重新转换成字符串用在报告中。

我们学到了很多格式化的方法和技巧。下面我们开始学习更加细致的数据清洗方法。我们将学习如何轻松发现数据中的坏种子（bad seed），以及如何处理它们。

7.2.3 找出离群值和不良数据

在数据集中寻找离群值和不良数据，可能是数据清洗中最难的任务之一，需要一段时间才

能做好。即使你对统计学有很深刻的理解，也完全了解离群值可能会对数据造成的影响，在研究这一话题时也要谨慎小心。

> 你要做的是清洗数据，不是处理数据或修改数据，所以在需要删除离群值或不良数据记录时，多花点时间思考如何处理这些数据。如果你剔除离群值使数据归一化，应该在最终结论中明确说明这一点。

第 9 章中我们会讲到更多寻找离群值的方法，但我们先考虑一些简单的方法，检查你的数据集中是否有不良数据。

判断数据有效性的第一个线索是数据来源。我们在第 6 章中说过，你需要仔细审核数据源，确保数据可信。你还需要咨询数据的采集方法，以及数据是否已经清洗过或处理过。

对于我们这里使用的数据样本来说，我们知道 UNICEF 调查有一套标准的问题格式。我们知道这些普查是定期进行的。我们还知道，他们在培训员工如何正确采访方面有一套标准规程。这些证据都说明，数据是一个好的样本，并不是事先选定的样本。相反，如果我们发现 UNICEF 只采访大城市的家庭，而忽略了农村人口，这可能会导致选择偏差或抽样误差。根据你的数据来源，你应该可以找出数据集可能具有的偏差。

> 你不可能每次都得到完美的数据。但你应该知道你的数据可能会有哪些取样偏差，确保你不会根据片面的数据集做出全面性的结论。

除了数据源和数据偏差，你还可以通过以下问题发现数据中可能存在的错误：“这些数据是否有不一致的地方？”发现错误数据的一个简单方法就是，查看数据值里是否有错误。例如，你可以浏览整个数据集，查看重要的数据值是否有缺失。你还可以浏览整个数据集，判断数据类型（例如整数、日期、字符串）是否正确匹配。在我们的数据集里尝试寻找缺失数据，看看里面是否有这些问题：

```
for answer in zipped_data[0]: ❶
    if not answer[1]: ❷
        print answer
```

❶ 遍历第一个数据条目的所有行。

❷ 测试某个值是否“存在”。我们要测试的值是元组的第二个元素，我们可以用 `if not` 语句来测试。

`if not` 语句

Python 可以用简单的 `if not` 语句来测试某个值是否存在。试着输入 `if not None: print True`。发生了什么？试着在 `if not` 后面加一个空字符串或零。发生了什么？

我们知道，我们的数据是字符串，而且不同于其他数据集，UNICEF 用空字符串来表示缺失数据（而不是 -- 等），所以我们可以通过测试字符串是否存在来测试值是否存在。

根据你的数据类型和数据集不同，你可能需要测试 `if x is None` 或 `if len(x) < 1`。你需要在代码的可读性和简洁之间做出平衡，保证代码具体而又明确。记得要遵循 Python 之禅（见 8.4 节）。

从代码的输出中可以看出，第一行没有明显缺失的数据。我们要如何测试整个数据集呢？

```
for row in zipped_data: ❶
    for answer in row: ❷
        if answer[1] is None: ❸
            print answer
```

❶ 这次我们不仅仅遍历第一行，而是遍历数据集的每一行。

❷ 我们删除了前面例子中的 `[0]`，因为我们要对每一行进行遍历。

❸ 作为例子，我们在这里测试是否有 `None` 类型的数据。我们可以知道是否有空的数据点，但无法知道是否有零或空字符串。

可以看出，整个数据集中没有明显缺失的数据，但我们来大致浏览其中一些数据，观察是否有不易发现的缺失数据。在前面的 `print` 语句中，你可能记得 NA 代表“不适用”（Not Applicable）。

虽然数据没有缺失，我们可能想知道到底有多少回答是 NA，或者某个问题的回答中 NA 的比例是否过高。如果样本量太小（大多数回答都是 NA），我们可能无法根据现有数据得出更一般性的结论。但如果大部分的回答都是 NA，我们可能会发现有趣的事情（为什么这个问题不适用于群体中的大多数人呢？）。

对于每一个具体的问题，我们来看一下回答是否以 NA 为主：

```
na_count = {} ❶

for row in zipped_data:
    for resp in row:
        question = resp[0][1] ❷
        answer = resp[1]
        if answer == 'NA': ❸
            if question in na_count.keys(): ❹
                na_count[question] += 1 ❺
            else:
                na_count[question] = 1 ❻

print na_count
```

❶ 定义一个字典，保存回答中包含 NA 的那些问题。将数据保存在哈希对象（比如字典）中，Python 可以快速方便地对数据进行查询。字典的键是问题，字典的值是包含 NA 的回答个数。

❷ 将元组第一部分的第二个元素（问题描述）保存在变量 `question` 中。第一个元素（`[0]`）是短标题，最后一个元素（`[2]`）是调查者的问题，有时这一项是缺失的。

❸ 利用 Python 的相等性测试找出值为 NA 的回答。如果注意到 NA 有多种写法，可以利用 `if answer in ["NA", "na", "n/a"]:` 语句判断具有相同含义的多个回答。
❹ 判断问题是否包含在字典的键中，从而判断问题是否已经包含在字典中。
❺ 如果问题已经包含在字典的键中，利用 Python 的 += 方法将字典的值加 1。
❻ 如果问题尚未包含在字典中，则将其添加到字典中，并将其对应的值设置为 1。

哇！我们的数据集中有好多 NA 回答啊。我们大约有 9000 行数据，其中一些问题有超过 8000 个 NA 回答。可能这些问题与调查的人群或年龄组无关，或者与特定的国家和文化没有太大关系。不管怎样，使用这些 NA 问题的意义不大，无法得出人口调查的任何一般性结论。

在判断数据集是否适用于你的研究目的时，寻找数据集中的 NA 是很有用的。如果发现你想要的问题有大量类似 NA 的回答，你可能需要继续寻找其他数据源，或者重新思考你的问题。

前面我们讲到了缺失数据，现在我们来看一下能否找到类型离群值（type outlier）。比如说，如果年份数据栏中出现了类似 `'missing'` 或 `'NA'` 这样的字符串，我们就说出现了类型离群值。如果只有几个数据的类型不匹配的话，我们可能需要处理离群值或几个不良数据。如果大部分数据的类型都不匹配的话，我们可能要重新思考是否要使用这些数据，或者找出这些数据看似“不良数据“的原因。

如果很容易就可以解释这些不一致的原因（比如这个回答只适用于女性，而调查样本中男女都有），那么我们就可以用这些数据。如果这些不一致没有明确的解释，而这个问题对我们的结果又很重要，我们需要继续研究当前数据集，或开始寻找能够对不一致作出解释的其他数据集。

在第 9 章中我们会讲到寻找离群值的更多内容，但现在我们来分析一下数据类型，看能否在当前数据集中找出明显的不一致之处。例如，我们应该检查一下，那些应该以数字作答的数据类型（比如出生年份）是否正确。

我们来看一下回答的类型分布。我们将用到前面计算 NA 回答数目的部分代码，但这次而我们要计算数据类型的数目：

```
datatypes = {} ❶

start_dict = {'digit': 0, 'boolean': 0,
              'empty': 0, 'time_related': 0,
              'text': 0, 'unknown': 0
              } ❷

for row in zipped_data:
    for resp in row:
        question = resp[0][1]
        answer = resp[1]
        key = 'unknown' ❸
        if answer.isdigit(): ❹
            key = 'digit'
        elif answer in ['Yes', 'No', 'True', 'False']: ❺
            key = 'boolean'
```

```
        elif answer.isspace(): ❻
            key = 'empty'
        elif answer.find('/') > 0 or answer.find(':') > 0: ❼
            key = 'time_related'
        elif answer.isalpha(): ❽
            key = 'text'
        if question not in datatypes.keys(): ❾
            datatypes[question] = start_dict.copy() ❿

        datatypes[question][key] += 1 ⓫

print datatypes
```

❶ 第一行代码初始化一个字典，因为用字典保存问题数据是一种快速、可靠的方法。

❷ 本行代码创建一个 `start_dict` 字典，用于检查数据集中每一个问题的数据类型是否相同。字典中包含所有可能的数据类型，方便我们对比。

❸ 这里我们将变量 `key` 设置为默认值 *unknown*。如果变量 `key` 在下面的 `if` 或 `elif` 语句中没有被修改的话，它的值还是 *unknown*。

❹ Python 字符串类有许多判断数据类型的方法。这里我们用的是 `isdigit` 方法：如果字符串中只包含数字，本行代码返回 `True`。

❺ 要判断数据是否与布尔逻辑相关，我们这里测试回答是否包含在一个由布尔回答组成的列表中，列表中包括 Yes/No 和 True/False。虽然我们可以创建一个更加全面的列表，但现在用这个列表就足够了。

❻ 如果字符串中只包含空格，Python 字符串类的 `isspace` 方法将返回 `True`。

❼ 字符串的 `find` 方法返回第一个匹配结果的索引编号。如果在字符串中没有找到匹配，则返回 `-1`。本行代码同时测试 `/` 和 `:`，这是时间字符串中的两个常用符号。这个检查并不全面，但可以作为初步检查。

❽ 如果字符串中只包含字母，字符串的 `isalpha` 方法将返回 `True`。

❾ 与计算 NA 回答数目的代码类似，我们这里判断问题是否包含在 `datatypes` 字典的键中。

❿ 如果问题没有包含在 `datatypes` 字典中，本行代码将问题添加到字典中，并将 `start_dict` 的副本作为对应的值。字典的 `copy` 方法为每一条数据创建一个独立的字典对象。如果将 `start_dict` 作为每一个问题对应的值，我们将会得到包含所有总数的一个字典，而不是每一个问题都对应一个新字典。

⓫ 将我们找到的键对应的值加 1。这样对于每一个问题和回答，我们对数据类型有了一个“猜测”。

从代码运行结果中已经可以发现不一致之处！有些问题以一种“类型”的回答为主，而其他问题则有许多种回答类型的猜测。我们可以从这里继续，因为它们只是粗略的猜测。

利用这一新信息的一种方法是，找到回答中大部分都是数字类型的问题，观察那些非数字的回答，看它们的值都是什么。我们认为可能会是 NA 或错误插入的值。如果这些值与我们关心的问题有关，我们可以将其归一化。一种做法是将 NA 或错误值替换为 None 或空值。如果你需要对列数据做统计分析的话，这一方法是很有用的。

在对数据集的后续处理过程中，你还会发现数据类型的离群值或 NA 回答。处理这些不一致数据的最佳做法取决于你对该话题和数据集的熟悉程度，也取决于你想要回答的问题。如果你要合并数据集，有时你可以舍弃那些离群值和不良数据，但注意不要忽视微小的趋势。

现在我们已经初步找出了数据集中的离群值及其规律，下面我们继续清除另一种不良数据——重复值，即使是我们自己也可能会创建重复值。

7.2.4 找出重复值

如果你要处理的是同一调查数据的多个数据集，或者是可能包含重复值的原始数据，删除重复数据是确保数据准确可用的重要步骤。如果你的数据集有唯一标识符，你可以利用这些 ID，确保没有误插入重复数据或获取重复数据。如果你的数据集没有索引，你可能需要找到判断数据唯一性的好方法（例如创建一个可索引的键）。

Python 内置库中有几个判断数据唯一性的好方法。我们首先介绍一些概念：

```
list_with_dupes = [1, 5, 6, 2, 5, 6, 8, 3, 8, 3, 3, 7, 9]

set_without_dupes = set(list_with_dupes)

print set_without_dupes
```

输出应该是这样的：

```
{1, 5, 6, 2, 6, 3, 6, 7, 3, 7, 9,}
```

这里发生了什么？集合（set）和 frozenset 都是 Python 的内置类型，输入一个可迭代对象（比如列表、字符串或元组），返回一个包含唯一值的集合。

要使用集合和 frozenset，输入的值需要是*可哈希的*（hashable）。对于可哈希的数据类型，我们可以使用哈希方法，得到的结果总是相同的。例如，我们可以认为代码中的每一个 `3` 都是完全相同的。

大部分 Python 对象都是可哈希的——只有列表和字典不是。对于任意可哈希类型（整数、浮点数、小数、字符串、元组等），我们可以用 `set` 创建集合。集合和 frozenset 的另一个巧妙之处在于，它们有一些可以快速比较的属性。我们来看几个例子：

```
first_set = set([1, 5, 6, 2, 6, 3, 6, 7, 3, 7, 9, 10, 321, 54, 654, 432])

second_set = set([4, 6, 7, 432, 6, 7, 4, 9, 0])

print first_set.intersection(second_set) ❶

print first_set.union(second_set) ❷

print first_set.difference(second_set) ❸
```

```
print second_set - first_set ❹

print 6 in second_set ❺

print 0 in first_set
```

❶ 集合的 intersection 方法返回两个集合的交集（即两个集合共有的元素）。内置的维恩图哦！

❷ 集合的 union 方法将第一个集合的值与第二个集合的值合并在一起。

❸ difference 方法给出的是第一个集合和第二个集合的差集。从下一行代码可以看出，运算顺序很重要。

❹ 用一个集合去减另一个集合，得出二者的差集。改变集合的顺序会改变结果（与数学中的减法类似）。

❺ in 判断元素是否包含在集合中（速度很快）。

你的输出应该是像这样的：

```
set([432, 9, 6, 7])
set([0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 321, 432, 654, 54])
set([1, 2, 3, 5, 321, 10, 654, 54])
set([0, 4])
True
False
```

在定义唯一数据集和集合对比方面，集合有许多实用的特性。在数据处理过程中，我们经常需要计算一系列值的最大值和最小值，或者需要唯一键的合并。集合可以帮我们完成这些任务。

除了集合，Python 还有一些库可以轻松测试唯一性。你可以用 numpy 库来测试唯一性，这是 Python 中一个强大的数学库，包含很多科学和统计的方法和类。与 Python 核心库相比，numpy 拥有出色的数组功能、数值计算功能和数学功能。numpy 数组还有一个好用的方法，叫作 unique。你可以这样安装 numpy：

```
pip install numpy
```

我们来看一下 numpy 库的 unique 的工作原理：

```
import numpy as np

list_with_dupes = [1, 5, 6, 2, 5, 6, 8, 3, 8, 3, 3, 7, 9]

print np.unique(list_with_dupes, return_index=True) ❶

array_with_dupes = np.array([[1, 5, 7, 3, 9, 11, 23], [2, 4, 6, 8, 2, 8, 4]]) ❷

print np.unique(array_with_dupes) ❸
```

❶ numpy 库的 unique 方法会保存索引编号。设置 return_index=True，返回的是由数组组成的元组：第一个是唯一值组成的数组，第二个是由索引编号组成的扁平化数组——只包含每一个数字第一次出现时的索引编号。

❷ 为了展示 numpy 的更多功能，本行代码创建了一个 numpy 矩阵。矩阵是由（长度相同的）数组组成的数组。
❸ unique 将矩阵转换成由唯一值组成的集合。

你的输出是这样的：

```
(array([1, 2, 3, 5, 6, 7, 8, 9]), array([ 0,  3,  7,  1,  2, 11,  6, 12]))
[ 1  2  3  4  5  6  7  8  9 11 23]
```

如果没有唯一键，你可以编写函数来创建唯一集合。写法和列表生成式一样简单。用 Python 集合在我们的数据集上试验一下。首先，我们观察数据集中哪些数据是唯一的，然后找出唯一数：

```
for x in enumerate(zipped_data[0]):
    print x

.....

(0, (['HH1', 'Cluster number', ''], '1'))
(1, (['HH2', 'Household number', ''], '17'))
(2, (['LN', 'Line number', ''], '1'))
(3, (['MWM1', 'Cluster number', ''], '1'))
(4, (['MWM2', 'Household number', ''], '17'))
(5, (['MWM4', "Man's line number", ''], '1'))
```

可以看出，每一行前五个元素可能包含了唯一标识符。假设我们对数据的理解是正确的，那么类群编号（Cluster number）、家庭编号（Household number）和男性家庭成员编号（Man's line number）三者应该是一个唯一组合。家庭成员编号（Line number）可能也是唯一编号。我们来看一下这是否正确：

```
set_of_lines = set([x[2][1] for x in zipped_data]) ❶

uniques = [x for x in zipped_data if not set_of_lines.remove(x[2][1])] ❷

print set_of_lines
```

❶ 首先，我们创建一个集合，里面包含调查中的所有家庭成员编号。家庭成员编号是每一个回答中的第三个元素，而编号值是里面第二个元素（x[2][1]）。我们使用列表生成式来提高代码的运行速度。
❷ set_of_lines 现在保存的是唯一键。我们可以利用集合对象的 remove 方法，判断数据集中每个键出现的次数是否多于一次。如果家庭成员编号是唯一的，那么每一个键只会删除一次。如果有重复值，remove 将会引发 KeyError，说明这个键已经不在集合中了。

嗯。运行代码时的确出现了错误，所以我们关于家庭成员编号是唯一的假设是错误的。如果仔细观察我们创建的集合，家庭成员编号似乎是从 1 到 16，然后依次重复。

> 你经常需要处理混乱的数据集，或者类似上面的数据集，没有明确的唯一键。遇到这种情况我们的建议是，找到唯一键，然后用这个唯一键来做对比。

创建唯一键的方法有很多。我们可以用采访的开始时间作为唯一键。但我们不确定UNICEF是否同时安排了多个调查组。如果是的话，我们可能会将事实上不是重复值的元素当作重复值删掉。我们可以用被采访人的出生日期和采访时间一起做唯一键，这样不太可能有重复值，但如果有字段缺失的话就麻烦了。

一种优雅的解决方法是，检查类群编号、家庭编号和家庭成员编号三者是否构成唯一键。如果是的话，我们可以将这个方法应用到整个数据集上——即使没有采访起止时间也可以。我们来试一下！

```
set_of_keys = set([
    '%s-%s-%s' % (x[0][1], x[1][1], x[2][1]) for x in zipped_data]) ❶

uniques = [x for x in zipped_data if not set_of_keys.remove(
    '%s-%s-%s' % (x[0][1], x[1][1], x[2][1]))] ❷

print len(set_of_keys) ❸
```

❶ 利用类群编号、家庭编号和家庭成员编号创建一个字符串，我们认为这三个编号的组合是唯一的。我们将三个编号用 - 隔开，这样方便区分。

❷ 利用 `remove` 方法重新创建我们用到的唯一键。这样会一个个删除所有数据，`uniques` 列表包含每一个唯一数据。如果有重复数据的话，代码还会抛出错误。

❸ 计算唯一键列表的长度。我们可以知道数据集中有多少个唯一值。

太好了！这一次没有报错。从列表的长度中可以看出，每一行都是唯一的。这也符合我们对这个数据集的预期，因为UNICEF在发布数据之前会做一些数据清洗工作，确保没有重复值。如果我们要将这些数据与其他UNICEF数据合并的话，我们可能需要将M添加到唯一键中，因为这是男性组的调查。然后我们可以对相同编号的家庭做交叉对照。

唯一键可能并不容易发现，这取决于你所用的数据。出生日期和地址可能是一个好的唯一键组合。两个24岁女性住在同一地方，出生日期又恰好相同的可能性是很小的，虽然也不是完全不可能，比如她们是住在一起的双胞胎！

讲完重复值，下面我们来讲模糊匹配，这是寻找重复值的好方法，尤其是杂乱的数据集。

7.2.5 模糊匹配

如果你要处理不止一个数据集，或者是未标准化的脏数据，可以用模糊匹配来寻找和合并重复值。模糊匹配可以判断两个元素（通常是字符串）是否“相同”。模糊匹配并不像自然语言处理或机器学习在处理大型语言数据集时那么深入，它可以帮我们判断“My dog & I”和“me and my dog”的意思相近。

模糊匹配有很多种做法。一个由SeatGeek（http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/）开发的Python库，使用很酷的内置方法来匹配多种场景的线上售票。安装方法如下：

```
pip install fuzzywuzzy
```

比如说你要处理一些脏数据。可能是输入时粗心，也可能是用户输入的，导致数据中包含

拼写错误和较小的语法错误或句法偏移。你要怎么处理这种情况呢？

```
from fuzzywuzzy import fuzz

my_records = [{'favorite_book': 'Grapes of Wrath',
               'favorite_movie': 'Free Willie',
               'favorite_show': 'Two Broke Girls',
              },
              {'favorite_book': 'The Grapes of Wrath',
               'favorite_movie': 'Free Willy',
               'favorite_show': '2 Broke Girls',
              }]


print fuzz.ratio(my_records[0].get('favorite_book'),
                 my_records[1].get('favorite_book')) ❶

print fuzz.ratio(my_records[0].get('favorite_movie'),
                 my_records[1].get('favorite_movie'))

print fuzz.ratio(my_records[0].get('favorite_show'),
                 my_records[1].get('favorite_show'))
```

❶ 这里我们用的是 `fuzz` 模块的 `ratio` 函数，接受两个字符串作比较。返回的是两个字符串序列的相似程度（一个介于 1 和 100 之间的值）。

根据我们自己对流行文化和英语的理解，这两组数据的爱好是相同的，但是二者拼写却不同。FuzzyWuzzy 可以帮我们处理这些无心之错。可以看出，`ratio` 的匹配得分相当高。这给了我们一些信心，相信这两个字符串是相似的。

我们再尝试另一个 FuzzyWuzzy 方法，看结果如何。为了简化问题并方便对比，我们采用的是相同的数据：

```
print fuzz.partial_ratio(my_records[0].get('favorite_book'),
                         my_records[1].get('favorite_book')) ❶

print fuzz.partial_ratio(my_records[0].get('favorite_movie'),
                         my_records[1].get('favorite_movie'))

print fuzz.partial_ratio(my_records[0].get('favorite_show'),
                         my_records[1].get('favorite_show'))
```

❶ 这里我们调用的是 `fuzz` 模块的 `partial_ratio` 函数，接受两个字符串作比较。返回的是匹配程度最高的子字符串序列的相似程度（一个介于 1 和 100 之间的值）。

哇，我们的得分更高了！ `partial_ratio` 函数可以比较子字符串，这样我们就不必担心某人漏掉一个字母（像上面书的例子）或拼写不同。这样所有字符串的匹配度都会更高。

> 如果你的数据有一些简单的不一致之处，这些函数都可以帮你找到不匹配的地方。但如果你的数据只有几个字符的差别，但在含义上却有很大不同，你可能想要同时测试相似性和不同之处。例如“does”和“doesn't”的含义截然不同，但在拼写上差别不大。在第一个 `ratio` 例子中，这两个字符串的得分不高，但子字符串却可以匹配。一定要了解你的数据及其内部的复杂性！

FuzzyWuzzy 还有其他很酷的功能。我们来研究其中几个，它们可能适用于你的数据清洗任务：

```
from fuzzywuzzy import fuzz

my_records = [{'favorite_food': 'cheeseburgers with bacon',
               'favorite_drink': 'wine, beer, and tequila',
               'favorite_dessert': 'cheese or cake',
              },
              {'favorite_food': 'burgers with cheese and bacon',
               'favorite_drink': 'beer, wine, and tequila',
               'favorite_dessert': 'cheese cake',
              }]

print fuzz.token_sort_ratio(my_records[0].get('favorite_food'), ❶
                            my_records[1].get('favorite_food'))

print fuzz.token_sort_ratio(my_records[0].get('favorite_drink'),
                            my_records[1].get('favorite_drink'))

print fuzz.token_sort_ratio(my_records[0].get('favorite_dessert'),
                            my_records[1].get('favorite_dessert'))
```

❶ 这里我们调用的是 `fuzz` 模块的 `token_sort_ratio` 函数，在匹配字符串时不考虑单词顺序。对于格式不限的调查数据来说，这个方法是很好用的，比如“I like dogs and cats”和“I like cats and dogs”的含义相同。每个字符串都是先排序然后再比较，所以如果包含相同的单词但顺序不同，也是可以匹配的。

从输出中可以看出，使用标记（这里是单词）有相当大的可能匹配顺序不同的单词。我们发现二者最爱的饮料相同，只是顺序不同。如果标记的顺序不会改变含义，我们就可以用这个方法。对于 SeatGeek 来说，“Pittsburgh Steelers vs. New England Patriots”与“New England Patriots vs. Pittsburgh Steelers”是完全相同的（只是主场优势不同）。

利用相同的数据，我们来看 FuzzyWuzzy 中另一个与标记有关的函数：

```
print fuzz.token_set_ratio(my_records[0].get('favorite_food'), ❶
                           my_records[1].get('favorite_food'))

print fuzz.token_set_ratio(my_records[0].get('favorite_drink'),
                           my_records[1].get('favorite_drink'))

print fuzz.token_set_ratio(my_records[0].get('favorite_dessert'),
                           my_records[1].get('favorite_dessert'))
```

❶ 这里我们调用的是 `fuzz` 模块的 `token_set_ratio` 函数，同样用的是标记方法，但比较的是标记组成的集合，得出两个集合的交集和差集。这个函数对排序后的标记尝试寻找最佳匹配，返回这些标记相似的比例。

这里可以看出，如果我们不知道数据集中的相似和不同之处，可能会有意想不到的副作用。其中一个答案的拼写是错误的。我们知道芝士蛋糕（cheesecake）和奶酪（cheese）是不同的东西，但利用标记集合方法，这两者却错误地匹配（false positive）了。我们没能正

确匹配包含芝士汉堡（cheeseburger）的回答，即使二者是完全相同的。你能用我们已经学过的另一个方法来做到这一点么？

FuzzyWuzzy 提供的最后一个匹配方法是 `process` 模块。如果你只有有限的几个选项和杂乱的数据，这个模块是很有用的。比如说回答只有 yes、no、maybe 和 decline to comment 四种。我们看一下如何对其匹配：

```
from fuzzywuzzy import process

choices = ['Yes', 'No', 'Maybe', 'N/A']

process.extract('ya', choices, limit=2) ❶

process.extractOne('ya', choices) ❷

process.extract('nope', choices, limit=2)

process.extractOne('nope', choices)
```

❶ 利用 FuzzyWuzzy 的 `extract` 方法，将字符串与可能匹配的列表依次比较。函数返回的是 `choices` 列表中两个可能的匹配。

❷ 利用 FuzzyWuzzy 的 `extractOne` 方法，返回 `choices` 列表中与我们的字符串对应的最佳匹配。

啊哈！给定几个单词，我们事先知道其“含义”相同，`process` 可以找出最佳猜测——在上面的例子中也是正确的猜测。`extract` 返回的是带有比例的元组，代码对回答字符串进行解析，并对其相似之处和不同之处作比较。`extractOne` 函数仅返回最佳匹配及其比例组成的元组。根据需求的不同，你可以选择 `extractOne` 仅找出最佳匹配，然后继续下一步。

现在你已经学过所有字符串匹配的内容了，下面我们来学习如何自己编写类似的字符串匹配函数。

7.2.6　正则表达式匹配

模糊匹配不一定总能满足你的需求。如果你只需要匹配字符串的一部分，应该怎么办？如果你只想匹配电话号码或电子邮件地址呢？在抓取数据时（我们将在第 11 章中学习），或编译多个来源的原始数据时，这些都是你会遇到的问题。正则表达式可以帮你解决上述大部分问题。

利用正则表达式，计算机可以对代码中的字符串或数据的模式进行匹配、查找或删除。开发人员往往对正则表达式怀有恐惧之心，因为它们可能会变得异常复杂、难以理解。但它们是很有用的，当你需要正则表达式来帮你解决问题时，一些简单的基础知识就可以帮你阅读、编写和理解它们。

虽然正则表达式的名声不太好，但其基本语法是相当简单易学的。表 7-1 给出了正则表达式的基础知识。

表7-1：正则表达式基础知识

<table>
<tr><th>字符/模式</th><th>文字说明</th><th>匹配实例</th></tr>
<tr><td>\w</td><td>匹配任意一个字母字符或数字字符，包括下划线</td><td>a、0 或 _</td></tr>
<tr><td>\d</td><td>匹配任意一个数字</td><td>1、2 或 4</td></tr>
<tr><td>\s</td><td>匹配任意一个空格字符</td><td>' '</td></tr>
<tr><td>+</td><td>匹配一个或多个（贪婪）模式或字符</td><td>\d+ 可以匹配 476373</td></tr>
<tr><td>\.</td><td>匹配 . 字符</td><td>.</td></tr>
<tr><td>*</td><td>匹配零个或多个（贪婪）字符或模式（与 if 的作用几乎相同）</td><td>\d* 可以匹配 03289 和 ''</td></tr>
<tr><td>|</td><td>匹配多个模式中的一个（类似 OR）</td><td>\d|\w 可以匹配 0 或 a</td></tr>
<tr><td>[] 或 ()</td><td>字符类（将你希望匹配的字符放在一个字符空间里）和字符组（将你希望匹配的字符放在一个组里）</td><td>[A-C] 或 (AlBlC) 都可以匹配 A</td></tr>
<tr><td>-</td><td>合并字符组</td><td>[0-9]+ 匹配 \d+</td></tr>
</table>

想查看更多实例，我们推荐将这个优秀的正则表达式备忘单（http://www.virtu-al.net/2009/04/30/powershell-regex-cheat-sheet/）添加到书签里。

作为一名 Python 开发人员，没有必要记住正则表达式的语法，但语法规范的正则表达式可以在很多方面帮到你。利用 Python 内置的正则表达式模块 re，你可以轻松查找基本的匹配和分组方法。

我们来看一下正则表达式的几个用法：

```
import re

word = '\w+' ❶
sentence = 'Here is my sentence.'

re.findall(word, sentence) ❷

search_result = re.search(word, sentence) ❸

search_result.group() ❹

match_result = re.match(word, sentence) ❺

match_result.group()
```

❶ 定义一个普通字符串的基本模式。这个模式可以匹配包含字母和数字、但不包含空格和标点的字符串。这个模式会一直匹配，直到无法匹配为止（+ 表示贪婪匹配！）。

❷ re 模块的 findall 方法可以找出这个模式在字符串中的所有匹配。成功匹配了句子中的每一个单词，但没有匹配句号。在这个例子中我们用的模式是 \w，所以不会匹配标点和空格。

❸ search 方法可以在整个字符串中搜索匹配。发现匹配后，则返回匹配对象。

❹ 匹配对象的 group 方法会返回匹配的字符串。

❺ match 方法只从字符串开头开始搜索。它的工作原理与 search 不同。

我们可以轻松匹配句子中的单词，根据我们的需要，我们还可以改变寻找匹配的方式。在上面的例子中我们看到，findall 返回的是所有匹配组成的列表。比如说你只想提取长文本中的网站。你可以利用正则表达式模式找到链接，然后利用 findall 从文本中提取出所有链接。或者你可以查找电话号码或日期。如果你能够将想要寻找的内容转换成简单的模式，并将其应用到字符串数据上，你就可以使用 findall 方法。

我们还用到了 search 和 match 方法，在上面的例子中二者返回的结果相同——它们匹配的都是句子中的第一个单词。我们返回的是一个匹配对象，然后可以利用 group 方法获取数据。group 方法还可以接受参数。用 .group(0) 试一下。发生了什么？你觉得 0 是什么意思？（提示：想想列表！）

search 和 match 实际上大不相同。我们再多看几个例子，才能发现二者的不同之处：

```
import re

number = '\d+' ❶
capitalized_word = '[A-Z]\w+' ❷

sentence = 'I have 2 pets: Bear and Bunny.'

search_number = re.search(number, sentence)

search_number.group() ❸

match_number = re.match(number, sentence)

match_number.group() ❹

search_capital = re.search(capitalized_word, sentence)

search_capital.group()

match_capital = re.match(capitalized_word, sentence)

match_capital.group()
```

❶ 定义一个数字模式。加号表示贪婪匹配，所以它会尽可能匹配所有数字，直到遇到一个非数字字符为止。

❷ 定义一个大写单词的匹配。这个模式使用方括号来定义更长模式的一部分。方括号的意思是，我们希望第一个字母是大写字母。后面紧跟着的是一个连续的单词。

❸ 我们这里调用 group 时发生了什么？可以看到，search 方法返回的是匹配对象。

❹ 你认为这里的结果会是什么？可能是数字，但实际上出现了错误。match 返回的是 None，而不是匹配对象。

现在我们可以更清楚地看到 search 和 match 的区别。利用 match 我们无法找到一个好的匹配，尽管事实上我们尝试的每一次搜索都有许多匹配。为什么会这样？前面说过，match 从字符串的开头开始搜索，如果没有找到匹配，它会返回 None。与此相反，search 会继续向后搜索，直到找到匹配为止。只有到达字符串末尾还没有找到匹配时，search 才会返回 None。如果你需要匹配以特定模式开头的字符串，用 match 比较好。如果你只想在字符串

中找到第一个匹配或任意匹配，最好选择 search。

关于正则表达式，这里还有一点需要注意：你注意到了吗？你预期找到的第一个大写单词是什么？是“I”还是“Bear”？为什么我们没有找到“I”？什么模式能够同时匹配二者？（提示：参考上面的表格，看看你都能使用哪些通配符变量！）

现在我们更多地了解了正则表达式的语法，以及 match、search 和 findall 的用法。下面我们看一下能否创建可以匹配多组的模式。在上面的例子中，我们只有一个模式组，所以我们对匹配结果调用 group 方法，只得到一个结果。但利用正则表达式你可以找到不止一个模式，你还可以给找到的匹配组起一个变量名，这样可以提高代码的可读性，还可以确保每组匹配正确。

让我们来试一下！

```
import re

name_regex = '([A-Z]\w+) ([A-Z]\w+)' ❶

names = "Barack Obama, Ronald Reagan, Nancy Drew"

name_match = re.match(name_regex, names) ❷

name_match.group()

name_match.groups() ❸

name_regex = '(?P<first_name>[A-Z]\w+) (?P<last_name>[A-Z]\w+)' ❹

for name in re.finditer(name_regex, names): ❺
    print 'Meet {}!'.format(name.group('first_name')) ❻
```

❶ 这里我们用的是相同的大写单词语法，用了两次，分别放在括号里。括号的作用是分组。

❷ 这里我们在 match 方法里用的模式包含多个正则表达式组。如果找到匹配的话，将返回多个匹配组。

❸ 对匹配结果调用 groups 方法，返回找到的所有匹配组构成的列表。

❹ 为各组命名可以让代码清晰明确。在这个模式中，第一组叫 first_name，第二组叫 last_name。

❺ finditer 的作用与 findall 类似，但返回的是一个迭代器（iterator）。利用这个迭代器，我们可以逐个查看字符串中的匹配。

❻ 利用我们学过的字符串格式化的知识，打印出我们的数据。这里我们从每个匹配中仅提取名字（first name）。

利用 ?P<*variable_name*> 为模式组命名，这样写出的代码更容易理解。从上面的例子中可以看出，创建两个（或多个）特定模式构成的组并找到匹配数据也是相当容易的。有了这些方法，在阅读别人写的（或者你六个月前写的）正则表达式时就不用费力猜测了。你能自己写一个正则表达式实例来匹配中间名缩写吗（如果有的话）？

利用强大的正则表达式，你可以快速识别字符串的内容，并轻松解析字符串中的数据。在解析特别杂乱的数据集时，比如网络抓取的数据，正则表达式的作用是无可替代的。想阅

读更多正则表达式的内容，我们推荐在 RegExr 网站（http://www.regexr.com/）上试用交互式正则表达式解析器，还可以通读免费的正则表达式教程（http://www.regular-expressions.info/tutorial.html）。

前面学过了这么多匹配方法，现在你可以轻松找出重复值。下面我们来看一下，在我们的数据集中遇到重复值时可以有哪些做法。

7.2.7　如何处理重复记录

根据你的数据状态，你可能希望合并重复记录。如果你的数据集只有重复行，那就无需担心数据存储的问题，这些数据已经包含在最终数据集内，你只需在清洗后的数据中删除或舍弃这些行即可。但如果你要合并不同的数据集，并希望保存每一条重复数据，那你需要思考用 Python 实现的最佳做法。

在第 9 章我们将会讲到合并数据的一般性方法，用到一些新库。但合并数据行还有简单的方法，和你解析数据的方法相同。如果你用 `DictReader` 提取数据的话，我们用一个例子来讲解这个方法。我们将合并男性数据集的一些数据行。这次我们希望基于家庭来合并数据，所以我们关注的是每一家的调查，而不是每个人的调查：

```
from csv import DictReader

mn_data_rdr = DictReader(open('data/unicef/mn.csv', 'rb')) ❶

mn_data = [d for d in mn_data_rdr]

def combine_data_dict(data_rows): ❷
    data_dict = {} ❸
    for row in data_rows:
        key = '%s-%s' % (row.get('HH1'), row.get('HH2')) ❹
        if key in data_dict.keys():
            data_dict[key].append(row) ❺
        else:
            data_dict[key] = [row] ❻
    return data_dict ❼

mn_dict = combine_data_dict(mn_data) ❽

print len(mn_dict)
```

❶ 我们用到 `DictReader` 模块，方便解析我们想要的所有字段。

❷ 我们定义了一个函数，还可以用在其他 UNICEF 数据集上。我们之所以将函数命名为 `combine_data_dict`，是因为函数的作用是将 `data_rows` 合并，然后返回一个字典。

❸ 定义一个新的数据字典，用于函数的返回值。

❹ 在前面的例子中我们利用类群编号、家庭编号和家庭成员编号创建一个唯一键，与之类似，本行代码也创建一个唯一键。“HH1”代表类群编号，“HH2”代表家庭编号。本行代码用这两个编号来表示唯一家庭。

❺ 如果已经添加过这个家庭，本行代码将当前数据行添加到数据列表中。

❻ 如果这个家庭尚未添加，本行代码新建一个列表，列表元素为当前数据行。

❼ 在函数末尾，返回新的数据字典。

❽ 现在传入数据行运行该函数，并将新生成的字典赋值给一个变量，以供后续使用。本行代码将新生成的字典命名为 `mn_dict`，我们可以利用这个字典来查看有多少个唯一家庭，以及每个家庭分别做了多少份调查。



如果函数结尾没有 `return` 的话，函数将会返回 `None`。在你开始编写自己的函数时，一定要注意返回值的错误。

我们找到了约 7000 个唯一家庭，这说明采访中有 2000 多的男性与其他男性属于同一家庭。本次采访每个家庭平均有 1.3 个男性。像这样的简单计算可以让我们对数据有更深入的了解，还可以帮我们思考数据的含义，并发现基于现有数据我们可以回答哪些问题。

7.3　小结

本章你学习了数据清洗的基础知识，以及在数据处理过程中数据清洗的重要性。你用到了一些 MICS 原始数据，并直接与这些数据进行了交互。现在你学会了观察数据，并能判断你可能会遇到的数据清洗问题。现在你还可以找到并删除错误数据和重复数据。

表 7-2 中详细描述了本章讲到的新概念和新库。

表7-2：Python编程的新概念和新库

概念/库	作用
列表生成式	利用迭代器、函数和 / 或 `if` 语句，可以方便快速地创建列表，用于进一步清洗和处理数据
字典的 `values` 方法	返回由字典的值组成的列表。在测试内部元素时很有用
`in` 和 `not in` 语句	测试内部元素。通常用于字符串或列表。返回一个布尔值
列表的 `remove` 方法	传入一个元素，删除列表中第一个匹配的元素。对于一个创建好的列表，如果你确切知道要删除的元素是什么，这个方法很有用
`enumerate` 方法	传入任意可迭代对象，返回一个由索引编号和对应的值组成的元组
列表的 `index` 方法	传入一个元素，返回列表中第一个匹配对象的索引编号。如果没有匹配的话，返回 `None`
字符串的 `format` 方法	可以将一系列数据轻松转换成易读的字符串。用 `{}` 作为数据占位符，传入数据的数量应与其数量相同。还可以用于字典的键名。有许多字符串格式化方法可用
字符串格式化（`.4f`、`.2%`）	数学标记（flag），可以将数字转换成简单易读的字符串
`datetime` 库的 `strptime` 和 `strftime` 方法	可以将 Python 日期对象轻松转换成字符串，以及将字符串转换成日期对象
`datetime` 库的 `timedelta` 对象	表示两个 Python 日期对象之间的时间差，或者用于修改日期对象（例如增加时间或减少时间）
`if not` 语句	测试其后语句的值是否非 `True`，与 `if` 语句的布尔逻辑相反

（续）

概念/库	作用
is 语句	测试第一个对象和第二个对象是否相同。在类型测试方面很有用（例如：is None、is list）。更多关于 is 的内容可参阅附录 E
字符串的 isdigit 和 isalpha 方法	测试字符串对象是否只包含数字或只包含字母。返回一个布尔值
字符串的 find 方法	传入一个子字符串，返回它在字符串中的索引位置。如果没有找到匹配，则返回 -1
Python 集合对象（https://docs.python.org/2/library/sets.html）	唯一元素的集合类。与列表的用法很相似，但没有重复值。有许多集合对比的方法（union、intersection、difference）
numpy 包（http://www.numpy.org/）	Python 基本数学库，是 SciPy 栈的一部分
FuzzyWuzzy 库（https://github.com/seatgeek/fuzzywuzzy）	用于字符串模糊匹配的库
正则表达式（https://en.wikipedia.org/wiki/Regular_expression）和 Python 的 re 库（https://docs.python.org/2/library/re.html）	可以用来编写模式并在字符串中寻找匹配

在下一章里，你还会继续练习这些数据清洗和数据分析技术，并利用这些技术更好地安排和重复数据清洗任务。我们会讲到数据归一化和标准化，以及如何将数据清洗脚本化并对脚本进行测试。

第8章

数据清洗：标准化和脚本化

你已经学习了数据的匹配和解析方法，以及如何寻找重复值，你已经开始探索数据清洗的奇妙世界。随着进一步理解你的数据集和你想要回答的问题，你需要考虑数据标准化和清洗自动化的问题。

本章我们将探索数据标准化的方法和时机，以及何时将数据清洗脚本化并对脚本进行测试。如果你管理的数据集是定期更新或新增数据的话，你需要使清洗过程尽可能高效清楚，这样你就可以将更多时间花在数据分析和撰写报告上。我们首先讲数据集的标准化（standardizing）和归一化（normalizing），以及如果数据集没有归一化应该怎么做。

8.1 数据归一化和标准化

数据集的标准化和归一化可能意味着利用当前数据计算新数据，也可能是对特定列或特定数据进行标准化或归一化，这取决于你的数据和所从事的研究类型。

从统计学的观点来看，归一化通常需要对数据集进行计算，使数据都位于一个特定的范围。比如说，你可能需要将测验成绩归一化到一定范围，这样你就可以准确查看成绩分布。你可能还需要对数据做归一化，以便准确查看百分位数，或不同群体（或世代）之间的百分位数。

假设你想查看某队在给定赛季得分的分布情况。你可能首先会将比赛分为赢、输、平三种情况。然后再进一步分为赢多少分、输多少分，等等。你还可以按比赛时长和每分钟得分数来分类。你可以访问所有这些数据集，现在你希望在球队之间进行对比。如果要对数据归一化，你可能会将总得分归一化到 0-1 区间。离群值（最高得分）将会接近于 1，较低得分将会接近于 0。然后你可以利用新数据的分布情况，查看有多少支球队的得分位于中游，在低分和高分区间是否有很多球队。你还可以找出离群值（比方说，如果大多数得分

都在 0.3 和 0.4 之间，那么你就知道，没在这个范围内的得分可能就是离群值）。

如果想对同样的数据做标准化，应该怎么做呢？举个例子，你可以将数据标准化，计算出每分钟的平均得分。然后你可以将平均得分作图，查看分布情况。哪些球队每分钟得分较高？有没有离群值？

你还可以计算标准差来查看分布情况。在第 9 章中我们会更全面地介绍标准化，但主要问题就是：数据的正常范围是什么？这个范围之外都有哪些数据？数据有没有什么规律？

可以看出，归一化和标准化是不同的。但二者通常都可以让研究人员或调查人员确定数据的分布，并明白该分布对后续研究或计算的含义。

数据标准化和归一化有时还需要删除离群值，这样你才能更好地发现数据的规律和分布。回头看前面的球队例子，如果你从整个联赛中删除顶级得分球员的得分，球队的成绩是否发生了巨大的变化？如果一名球员得到了所在球队一半的得分，那么回答是“是的”，这会使球队成绩发生巨大的变化。

与此类似，如果某支球队总是大比分胜出，从联赛数据中剔除这支队伍，可能会大幅改变平均得分及其分布情况。你可以使用归一化、标准化和剔除离群值的方法来帮你找到问题的答案，这取决于你要解决的问题。

8.2 数据存储

我们已经讲过几种数据存储的方法，现在有了可用的数据，我们先来复习一下这些方法。如果你正在使用数据库、知道预期的表格格式，并想要保存已经清洗过的数据，那么你应该继续使用第 6 章讲过的 Python 库来连接数据库并保存数据。对于这些 Python 库中的大部分库，你都可以使用游标直接向数据库提交。

> 我们强烈建议在数据库脚本中添加错误信息，在遇到网络故障或数据库故障时可以捕获这些错误信息。我们建议频繁向数据库提交，这样可以避免网络问题或延迟问题影响脚本的运行。

如果你用的是第 6 章中讲过的 SQLite 例子，你需要将新的干净数据保存到你的数据库中。我们来看一下如何做到这一点：

```
import dataset

db = dataset.connect('sqlite:///data_wrangling.db') ❶

table = db['unicef_survey'] ❷

for row_num, data in enumerate(zipped_data): ❸
    for question, answer in data: ❹
        data_dict = { ❺
            'question': question[1], ❻
            'question_code': question[0],
            'answer': answer,
            'response_number': row_num, ❼
```

```
            'survey': 'mn',
        }

    table.insert(data_dict) ❽
```

❶ 这里我们访问本地数据库。如果你将文件保存到其他目录，一定要修改文件路径，将其修改为数据库文件相对于当前目录的位置（例如，如果数据库文件保存在上层目录中：file:///../datawrangling.db）。

❷ 本行代码创建一个新表：`unicef_data`。我们知道很多 UNICEF 调查都有相同的规律，所以我们这个数据库名是没有歧义、可复用的。

❸ 我们希望保存所在的行编号，这样每个回答都有一个编号。本行代码用到了 `enumerate` 函数，这样在数据库中可以找到（每一行 / 每一个回答的）每一条数据（它们的共用一个行编号）。

❹ 我们知道，我们的数据被分割成元组，标题列表是元组的第一个元素，问题回答是元组的第二个元素。本行代码利用 `for` 循环解析其中包含的数据并进行存储。

❺ 每一个问题和回答在数据库中都有对应的条目，所以我们可以将每行（即每次采访）所有的回答合并在一起。本行代码创建一个字典，其中包含每次采访中每个回答的必要数据。

❻ 标题列表中第二个元素是问题的详细说明。本行代码将其保存为 `question`，并将 UNICEF 问题代码保存为 `question_code`。

❼ 为了记录每一行回答（或每一次采访），本行代码添加了 `enumerate` 函数得到的 `rownum`。

❽ 最后，利用新表的 `insert` 方法将新字典插入我们的数据库中。

我们希望将清洗过的数据保存到 SQLite 数据库中。我们创建了一个新的数据库，用到了 `enumerate` 函数，这样我们可以合并每一个回答（每一行）。如果我们要访问数据，可以访问新表，利用第 6 章学过的函数来查看所有的数据记录，并在需要时进行检索。

如果你希望将清洗过的数据导出到简单文件中，应该也很容易做到。我们来看一下：

```
from csv import writer


def write_file(zipped_data, file_name):
    with open(file_name, 'wb') as new_csv_file: ❶
        wrtr = writer(new_csv_file) ❷
        titles = [row[0][1] for row in zipped_data[0]] ❸
        wrtr.writerow(titles) ❹
        for row in zipped_data:
            answers = [resp[1] for resp in row] ❺
            wrtr.writerow(answers)


write_file(zipped_data, 'cleaned_unicef_data.csv') ❻
```

❶ `with...as` 的作用是将第一个函数的输出赋值给第二个变量名。本行代码将新文件 `open(file_name, 'wb')` 赋值给变量 `new_csv_file`。`'wb'` 的意思是以二进制模式写入。

❷ 初始化 CSV writer 对象，传入一个打开的文件，然后将 writer 对象赋值给 `wrtr` 变量。

❸ writer 对象需要数据列表来逐行写入，本行创建的是标题行的标题列表。长标题是元组第一部分的第二个元素，所以对应的代码是 `row[0][1]`。

❹ 用到了 writer 对象的 `writerow` 方法，将一个可迭代对象转换成一行逗号分隔的数据。本行代码写入的是标题行。
❺ 利用列表生成式提取出所有回答（元组的第二个元素）。
❻ 将利用列表生成式创建的所有列表或回答写入 CSV 数据文件。

这里我们用到了学过的语法，也用到了一些新语法。我们已经学过如何用 `with...as` 将简单函数的返回值赋值给一个变量名。这里我们希望将打开的文件赋值给 `new_csv_file` 变量。这种语法通常用于文件和其他 I/O 对象，因为 Python 执行完 `with` 代码块中的代码之后，它会自动关闭文件，这很棒！

此外，代码中我们用到了 CSV writer 对象，与 CSV reader 对象的用法类似。`writerow` 可以将包含所有数据列的列表写入到 CSV 文件中。



`writerow` 方法接受一个可迭代对象，所以一定要传入一个列表或元组。如果你传入一个字符串，那么看到一些有趣的 CSV（“l,i,k,e, ,t,h,i,s”）时不要惊讶。

我们还用到了列表生成式来创建标题列表和回答列表。由于我们不需要用这个函数生成一个新对象或修改过的对象，所以没有返回任何值。这个函数可以帮我们复习目前学过的许多概念。

如果你想用其他方法来保存数据，可以参考第 6 章给出的关于保存数据的建议。保存完清洗过的数据之后，你可以继续进行后面的清洗过程，并对数据进行分析。

8.3　找到适合项目的数据清洗方法

根据数据的可靠性，以及你分析数据的频率，你可以选择一种完全不同的数据清洗方式。如果你要处理的数据是非常杂乱的，或者有许多不同的来源，你可能无法准确地将清洗过程脚本化。



你需要分析将数据清洗完全脚本化所要付出的时间和精力，然后判断数据清洗自动化能否真正节省时间。

如果清洗过程特别繁琐，有很多步骤，你可能需要创建一个包含许多辅助脚本的仓库。这样即使你没有一个按顺序完成所有步骤的脚本，这个仓库也会为你提供许多函数，在整个数据处理过程中均可使用，还可以让你处理新数据时速度更快。举个例子，你有一些在列表或矩阵中搜索重复值的脚本，还有一些函数，可以从 CSV 导入或导出数据，或者格式化字符串和日期。对于这种方法，你可以随时导入这些函数并使用，可以在 IPython 或 Jupyter 中导入使用（我们会在第 10 章中学到），也可以在当前仓库的其他文件中导入使用。

如果你的清洗代码有固定的规律，不太可能发生变化，那么可以将整个清洗过程脚本化。

8.4 数据清洗脚本化

随着你的 Python 知识的逐步深化与丰富，你编写的 Python 代码也会逐渐变得复杂。现在你可以编写函数、解析文件、导入并使用多个 Python 库，甚至还可以存储数据。是时候开始将代码脚本化了。脚本化（scripting）的意思是，确定代码的结构，用于后续使用、学习和分享。

以 UNICEF 数据为例。我们知道，UNICEF 每隔几年会发布这些数据集，其中许多数据是不变的。调查不太可能发生较大变化——它是建立在多年经验的基础之上。考虑到这些事实，我们可以信任这些数据集有相当高的一致性。如果我们需要再次用到 UNICEF 数据，可能至少可以复用第一次写的脚本中的一部分代码。

目前我们代码的结构比较简单，也缺少代码文档。除了可读性较差外，这样的代码还很难复用。虽然现在我们可以看懂自己写的函数，但一年后我们还能准确地读懂并理解这些函数吗？我们把这些函数发给同事，他们能看懂我们的笔记吗？在我们对这些问题做出肯定的回答之前，最好一行代码也不要写。如果一年后我们无法读懂自己的代码，那么这些代码是没有任何用处的，当发布新报告时会有人（很可能是我们自己）重新写这些代码。

Python 之禅不仅适用于编写代码，还适用于组织代码，函数、变量和类的命名，等等。最好在选择命名上花点时间，判断哪些名字可以让你和他人都一目了然。注释和文档可以帮助理解，但代码本身也应该具有较强的可读性。

> 经常有人称赞 Python 是最容易读懂的语言之一，即使是看不懂代码的人也能读懂！保持代码语法简洁可读，这样解释代码功能的文档也不需要太长。

Python 之禅

Python 之禅（https://www.python.org/dev/peps/pep-0020/）总是非常值得参考的（还可以输入 `import this` 来轻松查看）。它的要点是，对于 Python（和许多语言）来说，尽可能保持明确、简洁和实用总是最好的。[1]

随着你编程水平的提高，明确和实用的含义可能会发生变化，但我们强烈建议你尽可能保持代码清晰、精确和简单。有时可能会使代码量变大或者运行时间变长，但随着经验地增长，你总会找到方法将代码写得既快速又清晰。

现阶段应该将代码写得尽可能清晰，这样以后回看代码时，你可以理解自己当时的意思。

通读 PEP-8 Python 风格指南（https://www.python.org/dev/peps/pep-0008/），并遵守里面的规则。有许多 PEP-8 的检查工具（linter），可以通读你的代码，并指出其中不符合 PEP-8 的地方。

注 1：中文版 Python 之禅可参见：https://wiki.python.org/moin/PythonZenChineseTranslate。——译者注

除了风格标准和用法，你还可以用检查工具评估代码的复杂度。有些是根据 McCabe 关于循环复杂度的理论和计算方法（https://en.wikipedia.org/wiki/Cyclomatic_complexity）来对代码进行分析。虽然不是每次都能将代码分割成简单的代码块，但你应该尽量将复杂任务拆分成更小、更简单的任务，降低代码复杂度，使代码更明确。

在使代码更加清晰明确的同时，另一个很有用的做法是，让可复用的代码块更加通用。但注意不要过于一般化（`def foo` 这样的定义毫无用处），但如果你创建通用的辅助函数，你将会经常用到它们（例如用一个列表创建 CSV，或者用包含重复值的列表创建一个集合），你的代码也会更加有序、简洁和简单。

> 如果所有报告都用相同的代码连接数据库或打开数据文件，你可以为此创建一个函数。编写通用的辅助函数，其目的是创建简单、可读、可用且不重复的代码。

表 8-1 汇总了一些编程的最佳实践，你可以在以后的编程中考虑这些做法。这些最佳实践并没有包含关于 Python 和编程的所有内容，但可以为今后的学习和编程打下良好的基础。

表8-1：Python编程最佳实践

实践	说明
文档	包括代码中的注释、函数说明和脚本说明，以及 README.md 文件和仓库中其他必要的说明文件
命名清晰	所有函数、变量和文件都应该有清晰的命名，从名字中就可以看出其内容或功能
语法正确	变量和函数应该遵守正确的 Python 语法（一般用小写字母，单词之间加下划线，对于类名采用驼峰式大小写（CamelCase，https://en.wikipedia.org/wiki/CamelCase），代码应遵守 PEP-8 标准
导入	只导入需要使用的内容，导入方式遵守 PEP-8 的原则
辅助函数	创建抽象的辅助函数，使代码变得清晰、可复用（例如，`export_to_csv` 函数将列表内容导入 CSV 文件）
仓库管理	用逻辑结构和层级结构管理仓库，共用的代码放在一起，符合一般的逻辑规律
版本控制	所有代码都应该有版本控制，这样你或你的同事可以创建新分支、尝试新特性，而不会影响仓库主分支的运行
快速，但是更要清晰	利用 Python 语法糖写出快速高效的代码，但当速度和清晰只能二选一时，选择清晰的代码
利用现成的库	当你想做点什么，而前人已经用 Python 做过了，不要重复造轮子。善于利用优秀的 Python 库，对这些库做贡献来帮助开源社区
代码测试	在适当可行的时候，为单个函数编写测试，并利用测试数据来测试代码
详实准确	在 `try` 代码块中正确地编写例外（exception），代码文档要详实，变量名要准确

为代码编写文档是编写脚本的一个重要步骤。正如 Eric Holscher（Python 主义者，Write the Docs 的创始人之一）恰如其分地总结（http://www.writethedocs.org/guide/writing/beginners-guide-to-docs/）：为代码编写文档的原因有很多，最重要的原因就是你可能会再次用到这些代码——或者其他人可能会阅读并使用这些代码，或者你想发布到 GitHub 上，或者你想在以后的面试中用到，或者你想将代码发给你母亲。无论什么原因，为代码编写

完备的文档，可以在未来减少数小时的痛苦。如果你是团队的一员，还会减少整个团队数百小时的痛苦。想到未来会有这些好处，现在值得花精力坐下来分析代码的用途，以及这么编写的原因。

类似 Read the Docs（https://readthedocs.org/）或者 Write the Docs（http://www.writethedocs.org/）之类的机构给出了许多好的建议和帮助，使编写文档变得更加轻松。一个好的经验做法是，在项目根目录里创建一个 README.md，简要说明代码的作用、安装方法和运行方法、基本要求以及在哪里可以找到更多信息。

> 有时在 README.md 里放一个简短的代码示例也是很有用的，这取决于用户（读者）与核心组件的交互次数多少。

除了 README.md 文件，你还需要添加代码注释。第 5 章中说过，注释可以是只给自己看的快速笔记，也可以是说明脚本和函数用法的长注释。

> Python 中各种注释的语法和用法在 PEP-350（https://www.python.org/dev/peps/pep-0350/）中有详细说明。遵循这些标准，任何人都可以轻松看懂你写的注释。

我们来尝试为之前的清洗代码编写文档。为了让我们编写文档的思路清晰，我们首先简要列出需要完成的任务。

- 从 UNICEF 数据文件中导入数据。
- 找到数据行对应的标题。
- 将我们可以读懂的标题与内置缩写标题正确匹配。
- 解析数据，检查是否有重复值。
- 解析数据，检查数据是否有缺失。
- 将同一家庭的多行数据合并。
- 保存数据。

上述任务基本上是按先后顺序排列的，列出这些任务，可以让我们在组织代码结构、编写脚本以及为新脚本编写文档时减轻一些痛苦。

我们要做的第一件事情，就是将本章和上一章写的所有代码块放到同一个脚本文件中。把它们放在一起之后，我们可以开始按照规则写出好代码。我们来看一下当前的脚本：

```
from csv import reader
import dataset

data_rdr = reader(open('../../../data/unicef/mn.csv', 'rb'))
header_rdr = reader(open('../../../data/unicef/mn_headers_updated.csv', 'rb'))

data_rows = [d for d in data_rdr]
header_rows = [h for h in header_rdr if h[0] in data_rows[0]]

all_short_headers = [h[0] for h in header_rows]
```

```
skip_index = []
final_header_rows = []

for header in data_rows[0]:
    if header not in all_short_headers:
        print header
        index = data_rows[0].index(header)
        if index not in skip_index:
            skip_index.append(index)
    else:
        for head in header_rows:
            if head[0] == header:
                final_header_rows.append(head)
                break

new_data = []

for row in data_rows[1:]:
    new_row = []
    for i, d in enumerate(row):
        if i not in skip_index:
            new_row.append(d)
    new_data.append(new_row)

zipped_data = []

for drow in new_data:
    zipped_data.append(zip(final_header_rows, drow))

# 检查数据是否有缺失

for x in zipped_data[0]:
    if not x[1]:
        print x

# 检查是否有重复值

set_of_keys = set([
    '%s-%s-%s' % (x[0][1], x[1][1], x[2][1]) for x in zipped_data])

uniques = [x for x in zipped_data if not
           set_of_keys.remove('%s-%s-%s' %
                              (x[0][1], x[1][1], x[2][1]))]

print len(set_of_keys)

# 保存到数据库

db = dataset.connect('sqlite:///../../data_wrangling.db')

table = db['unicef_survey']

for row_num, data in enumerate(zipped_data):
    for question, answer in data:
```

```
            data_dict = {
                'question': question[1],
                'question_code': question[0],
                'answer': answer,
                'response_number': row_num,
                'survey': 'mn',
            }

        table.insert(data_dict)
```

可以看出，大部分代码都是扁平的（flat），即没有重要性的嵌套关系。文件中大部分代码和函数都没有缩进或文档。代码本身不够抽象，变量名也不够清晰。我们从头开始解决这些问题。前两段代码重复。我们可以编写一个函数来代替：

```
def get_rows(file_name):
    rdr = reader(open(file_name, 'rb'))
    return [row for row in rdr]
```

有了这个函数，现在我们的文件就变短了。我们来看下一段代码是否还能进一步改进。

我们修改 header_rows 使其与 data_rows 里的标题对齐，花了不少时间，但现在已经不需要这段代码了。我们创建了 final_header_rows，里面的 header_rows 和 data_rows 已经匹配好了，所以我们无需担心二者不匹配的问题。我们可以删除这行代码。

14~27 行的作用是创建 final_header_rows 和 skip_index 两个列表。我们可以将这两个列表的用途总结一下，就是用于删除不匹配的元素，这样我们才能合并最终列表。我们把两个列表放在同一个方法中：

```
def eliminate_mismatches(header_rows, data_rows):
    all_short_headers = [h[0] for h in header_rows]
    skip_index = []
    final_header_rows = []

    for header in data_rows[0]:
        if header not in all_short_headers:
            index = data_rows[0].index(header)
            if index not in skip_index:
                skip_index.append(index)
        else:
            for head in header_rows:
                if head[0] == header:
                    final_header_rows.append(head)
                    break
    return skip_index, final_header_rows
```

现在我们已经将清洗脚本中的很多代码都合并成函数了。这有助于我们描述每一个函数的功能，编写代码文档，当需要修改代码时知道需要查看哪些内容。

我们继续阅读脚本，看能否找到更多需要修改之处。下一节代码似乎是用于创建合并后的数据集。我们可以将其拆分为两个函数：一个找出与标题匹配的数据行，另一个合并两个列表。我们也可以只用一个函数来创建合并后的数据。最终由你自己决定哪种方法更好。这里我们用的是一个函数外加一个简短的辅助函数，后面可能会再次用到：

```
def zip_data(headers, data):
    zipped_data = []
    for drow in data:
        zipped_data.append(zip(headers, drow))
    return zipped_data


def create_zipped_data(final_header_rows, data_rows, skip_index):
    new_data = []
    for row in data_rows[1:]:
        new_row = []
        for index, data in enumerate(row):
            if index not in skip_index:
                new_row.append(data)
        new_data.append(new_row)
    zipped_data = zip_data(final_header_rows, new_data)
    return zipped_data
```

有了这些新函数，我们可以保存代码、清洗变量名，还可以利用辅助函数将标题与数据行合并，并返回合并后的数据列表。代码更加清晰，分块也更加合理。我们继续将同样的逻辑应用到文件中的其他代码。我们来看一下最终结果：

```
from csv import reader
import dataset


def get_rows(file_name):
    rdr = reader(open(file_name, 'rb'))
    return [row for row in rdr]


def eliminate_mismatches(header_rows, data_rows):
    all_short_headers = [h[0] for h in header_rows]
    skip_index = []
    final_header_rows = []

    for header in data_rows[0]:
        if header not in all_short_headers:
            index = data_rows[0].index(header)
            if index not in skip_index:
                skip_index.append(index)
        else:
            for head in header_rows:
                if head[0] == header:
                    final_header_rows.append(head)
                    break
    return skip_index, final_header_rows


def zip_data(headers, data):
    zipped_data = []
    for drow in data:
        zipped_data.append(zip(headers, drow))
    return zipped_data
```

```
def create_zipped_data(final_header_rows, data_rows, skip_index):
    new_data = []
    for row in data_rows[1:]:
        new_row = []
        for index, data in enumerate(row):
            if index not in skip_index:
                new_row.append(data)
        new_data.append(new_row)
    zipped_data = zip_data(final_header_rows, new_data)
    return zipped_data


def find_missing_data(zipped_data):
    missing_count = 0
    for question, answer in zipped_data:
        if not answer:
            missing_count += 1
    return missing_count


def find_duplicate_data(zipped_data):
    set_of_keys = set([
        '%s-%s-%s' % (row[0][1], row[1][1], row[2][1])
        for row in zipped_data])

    uniques = [row for row in zipped_data if not
               set_of_keys.remove('%s-%s-%s' %
                                  (row[0][1], row[1][1], row[2][1]))]

    return uniques, len(set_of_keys)


def save_to_sqlitedb(db_file, zipped_data, survey_type):
    db = dataset.connect(db_file)

    table = db['unicef_survey']
    all_rows = []

    for row_num, data in enumerate(zipped_data):
        for question, answer in data:
            data_dict = {
                'question': question[1],
                'question_code': question[0],
                'answer': answer,
                'response_number': row_num,
                'survey': survey_type,
            }
            all_rows.append(data_dict)

    table.insert_many(all_rows)
```

现在我们有了许多不错的函数，却改变了程序的运行方式。如果现在运行这个脚本，一行代码都不会运行。只有一些写好的函数，却都没有被调用。

现在我们要在一个 `main` 函数中说明使用这些函数的方法。Python 开发者一般会将通过命令行运行的代码放到 `main` 函数里。下面我们添加 `main` 函数的代码，用于清洗数据集：

```
""" 这部分代码放在已写脚本的下面。"""

def main():
    data_rows = get_rows('data/unicef/mn.csv')
    header_rows = get_rows('data/unicef/mn_headers_updated.csv')
    skip_index, final_header_rows = eliminate_mismatches(header_rows,
                                                         data_rows)
    zipped_data = create_zipped_data(final_header_rows, data_rows, skip_index)
    num_missing = find_missing_data(zipped_data)
    uniques, num_dupes = find_duplicate_data(zipped_data)
    if num_missing == 0 and num_dupes == 0:
        save_to_sqlitedb('sqlite:///data/data_wrangling.db', zipped_data)
    else:
        error_msg = ''
        if num_missing:
            error_msg += 'We are missing {} values. '.format(num_missing)
        if num_dupes:
            error_msg += 'We have {} duplicates. '.format(num_dupes)
        error_msg += 'Please have a look and fix!'
        print error_msg


if __name__ == '__main__':
    main()
```

现在我们有了一个可以从命令行运行的可执行文件。运行此文件会发生什么？你会得到我们刚刚创建的错误信息，还是将数据保存到本地的 SQLite 数据库中？

使一个文件可以在命令行中运行

大多数可以在命令行中运行的 Python 文件都有一些相同的属性。它们一般都有一个 `main` 函数，里面再调用小型函数或辅助函数，和我们上面的清洗脚本类似。

`main` 函数一般会在文件的主缩进级别的代码块中进行调用。调用的语法是 `if __name__ == '__main__':`。这个语法用到了全局的**私有**变量（所以变量名两边才有双下划线），当你在命令行运行文件时会返回 `True`。

如果不是在命令行中运行脚本，那么 `if` 语句中的代码不会运行。如果我们将这些函数导入另一个脚本中，`__name__` 变量不等于 `'__main__'`，代码就不会运行。这是 Python 脚本常用的约定。

遇到任何错误，检查你的代码和上述代码是否完全相同，检查仓库中数据的文件路径是否正确，还要检查第 6 章创建的本地数据库的文件路径是否正确。

下面我们来为代码编写文档。我们要给函数添加一些文档字符串和行内注释，方便我们理解脚本中比较复杂的代码段，还要在脚本开头添加一大段说明文字，这些文字以后可以放到 README.md 文件中：

```
"""
Usage: python our_cleanup_script.py

This script is used to intake the male survey data from UNICEF
and save it to a simple database file after it has been checked
for duplicates and missing data and after the headers have been properly
matched with the data. It expects there to be a 'mn.csv' file with the
data and the 'mn_updated_headers.csv' file in a subfolder called 'unicef' within
a data folder in this directory. It also expects there to be a SQLite
file called 'data_wrangling.db' in the root of this directory. Finally,
it expects to utilize the dataset library
(http://dataset.readthedocs.org/en/latest/).

If the script runs without finding any errors, it will save the
cleaned data to the 'unicef_survey' table in the SQLite.
The saved data will have the following structure:
    - question: string
    - question_code: string
    - answer: string
    - response_number: integer
    - survey: string

The response number can later be used to join entire responses together
(i.e., all of response_number 3 come from the same interview, etc.).

If you have any questions, please feel free to contact me via ...
"""

from csv import reader
import dataset


def get_rows(file_name):
    """Return a list of rows from a given csv filename."""
    rdr = reader(open(file_name, 'rb'))
    return [row for row in rdr]


def eliminate_mismatches(header_rows, data_rows):
    """
    Return index numbers to skip in a list and final header rows in a list
    when given header rows and data rows from a UNICEF dataset. This
    function assumes the data_rows object has headers in the first element.
    It assumes those headers are the shortened UNICEF form. It also assumes
    the first element of each header row in the header data is the
    shortened UNICEF form. It will return the list of indexes to skip in the
    data rows (ones that don't match properly with headers) as the first element
    and will return the final cleaned header rows as the second element.
    """
    all_short_headers = [h[0] for h in header_rows]
```

```
    skip_index = []
    final_header_rows = []

    for header in data_rows[0]:
        if header not in all_short_headers:
            index = data_rows[0].index(header)
            if index not in skip_index:
                skip_index.append(index)
        else:
            for head in header_rows:
                if head[0] == header:
                    final_header_rows.append(head)
                    break
    return skip_index, final_header_rows


def zip_data(headers, data):
    """
    Return a list of zipped data when given a header list and data list. Assumes
    the length of data elements per row and the length of headers are the same.

    example output: [(['question code', 'question summary', 'question text'],
                       'resp'), ....]
    """
    zipped_data = []
    for drow in data:
        zipped_data.append(zip(headers, drow))
    return zipped_data


def create_zipped_data(final_header_rows, data_rows, skip_index):
    """
    Returns a list of zipped data rows (matching header and data) when given a
    list of final header rows, a list of data rows, and a list of indexes on
    those data rows to skip as they don't match properly. The function assumes
    the first row in the data rows contains the original data header values,
    and will remove those values from the final list.
    """
    new_data = []
    for row in data_rows[1:]:
        new_row = []
        for index, data in enumerate(row):
            if index not in skip_index:
                new_row.append(data)
        new_data.append(new_row)
    zipped_data = zip_data(final_header_rows, new_data)
    return zipped_data


def find_missing_data(zipped_data):
    """
    Returns a count of how many answers are missing in an entire set of zipped
    data. This function assumes all responses are stored as the second element.
    It also assumes every response is stored in a list of these matched question,
    answer groupings. It returns an integer.
```

```
    """
    missing_count = 0
    for response in zipped_data:
        for question, answer in response:
            if not answer:
                missing_count += 1
    return missing_count


def find_duplicate_data(zipped_data):
    """
    Returns a list of unique elements and a number of duplicates found when given
    a UNICEF zipped_data list. This function assumes that the first three rows of
    data are structured to have the house, cluster, and line number of the
    interview and uses these values to create a unique key that should not be
    repeated.
    """

    set_of_keys = set([
        '%s-%s-%s' % (row[0][1], row[1][1], row[2][1])
        for row in zipped_data])

    #TODO: this will throw an error if we have duplicates- we should find a way
    #around this
    uniques = [row for row in zipped_data if not
               set_of_keys.remove('%s-%s-%s' %
                                  (row[0][1], row[1][1], row[2][1]))]

    return uniques, len(set_of_keys)


def save_to_sqlitedb(db_file, zipped_data, survey_type):
    """
    When given a path to a SQLite file, the cleaned zipped_data, and the
    UNICEF survey type that was used, saves the data to SQLite in a
    table called 'unicef_survey' with the following attributes:
        question, question_code, answer, response_number, survey
    """
    db = dataset.connect(db_file)

    table = db['unicef_survey']
    all_rows = []

    for row_num, data in enumerate(zipped_data):
        for question, answer in data:
            data_dict = {
                'question': question[1],
                'question_code': question[0],
                'answer': answer,
                'response_number': row_num,
                'survey': survey_type,
            }
            all_rows.append(data_dict)

    table.insert_many(all_rows)
```

```python
def main():
    """
    Import all data into rows, clean it, and then if
    no errors are found, save it to SQlite.
    If there are errors found, print out details so
    developers can begin work on fixing the script
    or seeing if there is an error in the data.
    """

    #TODO: we probably should abstract these files so that we can pass
    # them in as variables and use the main function with other surveys
    data_rows = get_rows('data/unicef/mn.csv')
    header_rows = get_rows('data/unicef/mn_updated_headers.csv')
    skip_index, final_header_rows = eliminate_mismatches(header_rows,
                                                         data_rows)
    zipped_data = create_zipped_data(final_header_rows, data_rows, skip_index)
    num_missing = find_missing_data(zipped_data)
    uniques, num_dupes = find_duplicate_data(zipped_data)
    if num_missing == 0 and num_dupes == 0:
        #TODO: we probably also want to abstract this
        # file away, or make sure it exists before continuing
        save_to_sqlite('sqlite:///data_wrangling.db', zipped_data, 'mn')
    else:
        #TODO: eventually we probably want to log this, and
        # maybe send an email if an error is thrown rather than print it
        error_msg = ''
        if num_missing:
            error_msg += 'We are missing {} values. '.format(num_missing)
        if num_dupes:
            error_msg += 'We have {} duplicates. '.format(num_dupes)
        error_msg += 'Please have a look and fix!'
        print error_msg


if __name__ == '__main__':
    main()
```

现在我们的代码文档更详细、结构更合理，还有许多可复用的函数。对于我们的第一个脚本来说，这是一个很好的开始。利用这些代码，希望我们可以导入许多 UNICEF 数据！

> 我们还在代码里添加了许多“TODO”（待办）的注释，这样我们以后可以继续完善脚本。你认为哪个问题是最迫切需要解决的？为什么？你能尝试解决这个问题吗？

我们只用了一个文件来运行代码。但随着代码量的增加，你的仓库也会变得越来越复杂。在初期就要思考你可能需要向仓库中添加的内容，这一点是很重要的。代码和代码结构很相似。如果你认为这个仓库可能的用途不仅仅是解析 UNICEF 数据，你的代码结构可能会大不相同。

为什么会这样？首先，你可能需要将数据保存在一个单独的文件中。事实上，根据你的仓

库未来的复杂程度，你可能需要在不同的文件夹中使用不同的数据解析方法和清洗方法。

> 在初期不必过分担心这些决策。随着你 Python 编程水平的提高和对数据集的理解进一步加深，你会更清楚地认识到应该从哪里开始。

在仓库的结构中，经常会有一个名为 utils 或 common 的文件夹，你可以在里面保存代码之间共享的脚本。许多开发者将数据库连接脚本，常用的 API 代码和通信或 email 脚本等保存在这样的文件夹中，方便导入其他脚本中。

你可能创建了多个目录来保存项目的不同内容，具体取决于仓库的管理结构。其中一个目录只和 UNICEF 数据有关。另一个目录可能包含网络抓取脚本或最终报告代码。如何组织仓库的结构由你自己决定。永远保持清晰、明确、有序。

如果你最后不得不重新组织仓库结构，那么在开始时就尽可能保持仓库有序，后面就不会太过痛苦。相反，如果你的仓库里都是 800 行的文件，而且没有清晰的文档，那么你要做的事情就很多了。最好的经验做法是最开始给出结构框架，随着仓库内容的增加和变化对结构进行临时调整。

除了良好的文件结构，保持目录、文件、函数和类的命名清晰明确也是很有用的。在 utils 文件夹中可能有多个文件。如果你将其命名为 utils1、utils2 等，你可能需要打开文件才能知道它们的具体内容。但如果你将其命名为 email.py、database.py、twitter_api.py 等，文件名本身就包含了更多信息。

在代码中尽量保持明确，对长期而成功的 Python 数据处理事业是一个良好的开端。我们思考一下仓库的结构，看如何找到相应的文件：

```
data_wrangling_repo/
|-- README.md
|-- data_wrangling.db
|-- data/
|   `-- unicef/
|       |-- mn.csv
|       |-- mn_updated_headers.csv
|       |-- wm.csv
|       `-- wm_headers.csv
|-- scripts/
|   `-- unicef/
|       `-- unicef_cleanup.py (本章的脚本)
`-- utils/
    |-- databases.py
    `-- emailer.py
```

我们还没有编写 databases 或 emailer 文件，但我们或许应该这么做。我们还可以向文件结构中添加哪些内容？我们在仓库中创建了两个不同的 unicef 文件夹，你认为这么做的原因是什么？开发者是否应该将数据文件和脚本文件分开保存？

> 你的项目文件结构可能和这个类似，但要记得，数据通常都不保存在仓库中。将项目的数据文件保存在共享文件服务器或本地网络的其他位置。如果你是独立开发，一定要在其他地方备份。不要将这些大文件提交到你的仓库中。这样不仅会在需要在新设备上查看仓库时降低工作效率，而且也不是管理数据的好方法。

我们也建议不要将 db 文件或任何 log、config 文件提交到仓库中。仓库结构应尽可能实用。你总是可以将预期的文件结构添加到 README.md 文件中，并详细说明去哪里获取数据文件。

Git 和 .gitignore 文件

如果你还没有用 Git（https://git-scm.com/）做版本控制的话，学完本书就会用了！版本控制可以让你创建仓库来管理和修改代码，并将其分享给团队或其他同事。

在第 14 章中我们将会深入讲解 Git，但现在我们在讨论仓库结构，希望重点说一下 .gitignore 文件（https://github.com/github/gitignore）。.gitignore 文件的作用是，让 Git 忽略某些文件，不要将这些文件上传到仓库中。这个文件使用简单模式来匹配文件名，与我们在第 7 章中学过的正则表达式类似。

在我们的仓库结构中，我们可以用一个 .gitignore 文件，这样 Git 就不会将任何数据文件提交到仓库中。然后我们可以在 README.md 中说明仓库的结构，给出获取数据文件的联系信息。这样我们的仓库就比较简洁，且易于下载，还可以保持良好的代码结构。

创建一个符合逻辑的仓库结构，并添加 README.md 和 .gitignore 文件，可以保持模块化代码的项目文件夹有序，并避免将大型数据文件或可能敏感的数据（数据库或登录数据）放在仓库中。

8.5 用新数据测试

前面我们已经学习了编写文档、代码脚本化、组织代码结构，现在我们应该编写一些测试，或者用新数据来测试。这可以帮我们检查代码的运行是否正确，是否与与预期相同，还可以明确代码的含义。我们将数据清洗脚本化的原因之一就是我们可以复用这些代码，因此用新数据测试可以证明我们在代码标准化上花的时间和精力是值得的。

测试刚写过的脚本的一种方法是，我们能否将其轻松应用于在 UNICEF 网站上找到的相似数据。我们来看一下。你应该已经从本书仓库（https://github.com/jackiekazil/data-wrangling）中下载了 wm.csv 和 wm_headers.csv 两个文件。这两个文件是津巴布韦 UNICEF 数据中的女性调查数据。

我们尝试在脚本中使用这些文件，替换男性调查数据。要做到这一点，我们只需修改清洗脚本中的两个文件名，将其修改为两个妇女调查数据文件的路径。我们还应该将调查类型修改为 `'wm'`，这样我们才能区分两个数据集中的数据。

> 女性数据集要比男性的大得多。如果你有未保存的数据，我们建议先保存数据，关闭其他程序，然后再进行下一步。关于这一点，可以思考一下如何在脚本中改善内存使用。

我们来看一下能否成功导入数据：

```
import dataset

db = dataset.connect('sqlite:///data_wrangling.db')

wm_count = db.query('select count(*) from unicef_survey where survey="wm"') ❶

count_result = wm_count.next() ❷

print count_result
```

❶ 我们用的是直接查询，可以快速查看 `survey='wm'` 的行数。这应该只包括我们将类型设为 `'wm'` 后第二次运行的数据行。

❷ 读取查询结果，利用查询响应的 `next` 方法提取出第一个结果。我们用的是 `count`，所以我们应该只得到一个结果。

我们从女性数据集中成功导入了超过 300 万个问题和回答。我们的脚本是有效的，我们可以看到输出结果！

利用相似数据测试脚本是确保脚本按预期运行的一种方法。从中还可以发现，你的脚本通用性很高，可以复用。但测试代码还有很多其他方法。Python 有不少很好的测试库，可以帮你编写测试脚本和应用测试数据（甚至测试 API 响应），从而保证代码正常运行。

Python 标准库中有一些内置的测试模块。`unittest` 库（https://docs.python.org/2/library/unittest.html）可以为 Python 代码做单元测试。它有一些好用的内置类，利用 `assert` 语句来测试代码是否正常运行。如果想为代码编写单元测试，我们可以写一个测试，判断 `get_rows` 函数返回的是不是一个列表。我们还可以判断列表长度和文件的数据行数是否相同。我们可以用这些断言测试每一个函数。

另一个流行的 Python 测试框架是 `nose` 库（https://nose.readthedocs.org/en/latest/）。`nose` 是一个非常强大的测试框架，额外插件（https://nose.readthedocs.io/en/latest/plugins/builtin.html）和配置可以提供很多功能选项。如果你的仓库很大，许多开发者负责同样的代码，有不同的测试需求，那么这个库是很好用的。

不知道先用哪一个？那么 `pytest` 库（http://pytest.org/latest/）可能适合你。你可以用任意一种风格编写测试，还可以在需要时换用另一种风格。它的社区也相当活跃，里面有许多演讲和教程（http://docs.pytest.org/en/latest/talks.html#talks-and-blog-postings），所以如果你想深入了解之后编写自己的测试，那么可以先用这个库。

> 通常来说，测试集的结构是在每一个模块下有一个测试文件（对于我们当前的仓库结构来说，我们会在每一个目录下放一个测试文件，除了数据文件夹和配置文件夹）。有些人为文件夹中的每一个 Python 文件编写一个测试文件，所以很容易判断每项测试针对的是哪一个具体文件。其他人将测试放在一个单独的目录中，其结构与 Python 文件结构相同。

无论你选择哪种测试风格或文件结构，一定要确保前后一致、风格明确。这样你就会知道在哪里可以找到测试文件，你（和其他人）也可以在必要时运行这些测试代码。

8.6 小结

本章我们学习了数据标准化的一些基本知识，还有何时适合做数据归一化或删除离群值。你可以将干净的数据（来自第 6 章）导入到数据库或本地文件中，并且你开始为那些重复性过程编写了条理更加清晰的函数。

此外，你还学习了用嵌套文件夹和正确命名的文件来组织 Python 仓库结构，开始编写文档并分析代码。最后，你对测试和编写测试的一些工具有了基本的了解。

表 8-2 列出了本章讲到的 Python 概念。

表8-2：Python编程的新概念和新库

概念/库	作用
数据库 `insert` 方法	利用 `insert` 命令可以将数据轻松保存到 SQLite 数据库中
CSV writer 对象	利用 `csv` 库的 `writer` 类，可以将数据保存到 CSV 文件中
Python 之禅（`import this`）	像 Python 程序员一样写代码和思考的哲学
Python 最佳实践	作为一名新的 Python 开发者，应该遵循的最佳实践的基本框架
Python 命令行运行（`if __name__ == '__main__':`）	利用这个代码块对脚本进行格式化，可以在命令行中运行 `main` 函数
TODO 标记	利用 TODO 注释，很容易发现下一步需要对脚本做哪些改进
Git（https://git-scm.com/）	帮助记录代码变化的版本控制系统。对于你想要部署的代码或想要与他人共享的代码，这一点是绝对必要的，而且对于本地的个人项目也是非常有用的。第 14 章会介绍 Git 的更多内容

在下一章里，我们将学习数据分析，你将会继续练习数据清洗和数据分析的方法，并利用这些方法来分析新的数据集。

第 9 章

数据探索和分析

既然已经花费了一些时间获取和清洗数据，你可以开始做数据分析了！不要对结果有太多期望，这对于你的数据探索过程来说很重要。你的问题可能对于某个答案来说太宽泛，也可能没有结论性的答案。回想一下你在第一节自然科学课程中学到的有关假设和结论的知识。最好用同样的方法来进行数据探索，并且要理解，在数据分析中你可能不会得到一个清晰的结论。

尽管如此，只是去探索数据并发现数据中没有趋势或者趋势不符合预期，这就很有趣。如果一切都如我们所愿，数据处理会变得有些无聊。我们已学会少一点期待，多一点探索。

> 当你开始分析和探索数据时，可能会意识到需要更多的或不同种类的数据。在你更深入地定义想要回答的问题，并检验数据告诉你什么的过程中，这是很常见的一种情况，你需要接受。

现在也非常适合回顾一下你最初发现数据集时所提出的问题。你想知道什么？是否还有其他相关的问题有助于你的探索？这些问题可能会指出方向，告诉你在哪能找到故事。即使没有，这些问题也会指引你发现另外一些有趣的问题。即使你不能回答最初的问题，也能够对话题有更深入的了解，并发现新的问题去探索。

在这一章，我们会学习一些新的用于数据探索和分析的 Python 库，并且继续应用我们在前两章学到的清洗数据的知识。我们会学习如何合并数据集，探索数据，得到有关数据集中关系的统计学结论。

9.1　探索数据

在前两章你已经学习了解析和清洗数据，想必已经很熟悉用 Python 来与数据交互了。现在

我们要使用 Python 更加深入地探索数据。

首先我们要安装将用到的 Python 库 agate（http://agate.readthedocs.org），它可以帮助我们发现数据的一些基本特征。agate 是一个数据分析库，由 Christopher Groskopf（https://github.com/onyxfish）编写。Christopher 是一位拥有高超技术水平的数据记者和 Python 开发者，而 agate 库会帮助我们了解数据。使用 pip 安装这个库：

```
pip install agate
```

这一章中的代码与 agate 1.2.0 版本兼容。因为 agate 是一个相对较新的 Python 库，所以随着库的成熟，其中的一些功能是有可能发生改变的。为确保安装的是指定版本的库，你可以使用 pip 设置版本。对本书来说，你可以使用：pip install agate==1.2.0 来安装 agate。我们同样推荐你测试最新的版本，并随时了解书中用到的库的最新代码变化。

我们想要探索 agate 库的一些特性。为了达到这个目的，我们将会使用从 UNICEF 年报得到的关于童工雇用的数据（http://data.unicef.org/child-protection/child-labour.html）。

9.1.1　导入数据

首先，来看一下我们的第一个数据集——UNICEF 的童工汇总数据。我们下载的数据是一个 Excel 文件，包含全世界的童工雇用率列表。我们可以使用学到的关于 Excel 的知识以及从第 4 章和第 7 章学到的数据清洗技术，将原始数据转化为 agate 库所接受的格式。

在处理 Excel 表单时，我们推荐你用喜欢的 Excel 查看器打开表单文件。这样我们更容易比对 Python“看到”的结果和我们从表单中看到的结果，这有助于数据的查找和提取。

首先，我们导入可能会用到的 Python 库，并且将 Excel 文件加载到 xlrd 中。

```
import xlrd
import agate

workbook = xlrd.open_workbook('unicef_oct_2014.xls')

workbook.nsheets

workbook.sheet_names()
```

现在将 Excel 中的数据加载到变量 workbook 中了。这个工作表包含一个表单，叫作 Child labour。

如果你在 IPython 终端中运行上面这段代码（我们推荐使用 IPython，因为你会看到更多的输出），应该看到下面的输出：

```
In [6]: workbook.nsheets
Out[6]: 1

In [7]: workbook.sheet_names()
```

```
Out[7]: [u'Child labour ']
```

选择这个表单，这样你可以将其导入 agate 库。根据 agate 库的文档（http://agate.readthedocs.io/en/latest/tutorial.html），agate 库可以使用一个标题列表、一个各数据列的类型列表和一个数据读取器（或可迭代的数据列表）来导入数据。所以，我们需要数据的类型，这样才可以顺利地从表单中导入数据到 agate 库。

```
sheet = workbook.sheets()[0]

sheet.nrows ❶

sheet.row_values(0) ❷

for r in range(sheet.nrows):
    print r, sheet.row(r) ❸
```

❶ nrows 标识表单中共有多少列。

❷ row_values 允许选取一行数据，并且展示这行的值。上面这种情况下，会展示表单数据的标题，因为标题是 Excel 文件的第一行。

❸ 通过使用 range 和 for 遍历每一行数据，我们可以像 Python 一样查看每一行数据。表单的 row 方法会返回数据的一些信息和每行数据的类型。

我们从第 3 章了解到，csv 库接受一个打开的文件，并且将其转化为一个迭代器（iterator）。迭代器是一个可以迭代或遍历的对象，每一次返回其对应的值。在代码中，迭代器在展开数据方面比列表更高效，拥有速度和性能的优势。

> 因为我们正在处理的数据集很小，所以可以创建一个列表，用其代替迭代器来传参。大多数需要迭代器的库可以处理任何可迭代对象（如列表）。通过这种方式，我们依然遵从 xldr 和 agate 所期待的方式。

首先，让我们获取每一列的标题。在之前的输出中，我们可以看到标题在第 4 行和第 5 行。可以使用 zip 来合并标题行：

```
title_rows = zip(sheet.row_values(4), sheet.row_values(5))

title_rows
```

现在可以看到变量 title_rows 的值为：

```
[('', u'Countries and areas'),
(u'Total (%)', ''),
('', ''),
(u'Sex (%)', u'Male'),
('', u'Female'),
(u'Place of residence (%)', u'Urban'),
('', u'Rural'),
(u'Household wealth quintile (%)', u'Poorest'),
('', u'Second'),
('', u'Middle'),
('', u'Fourth'),
('', u'Richest'),
```

```
(u'Reference Year', ''),
(u'Data Source', '')]
```

同时使用两行信息（第 4 行和第 5 行），会保留如果我们仅使用其中一行信息丢失的那部分信息。这是个很好的选择，我们还可以再花点时间来改进这一点，但是，对于最初的数据探索来说，这是一个很好的开始。标题数据当前是一个元组列表。我们知道 agate 库希望得到一个元组列表，其中第一个值为标题的字符串，这样我们需要将标题转换成字符串列表。

```
titles = [t[0] + ' ' + t[1] for t in title_rows]

print titles

titles = [t.strip() for t in titles]
```

在这段代码里面，我们使用了两个列表生成式。在第一个中，我们把 title_rows 列表传入，它是一个元组列表。在这些元组中，存有 Excel 文件中标题行的内容字符串。

第一个列表生成式使用了元组的两个部分（通过元组索引）来拼接一个字符串。我们将每一个元组的值合并在一起，出于可读性的考虑，使用 ' ' 做间隔。现在我们的标题列表只由字符串组成——原来的元组不见了！我们使得标题变得有一些复杂，因为并不是每一个元组都有两个值。通过添加空格分隔，我们创建了一些以空格为开始的标题，像 'Female'。

为了删除起始位置的空格，在第二个迭代器中，我们使用 strip 字符串方法，这个方法会移除字符串最开始和最后的空格。现在标题变量有了整洁的字符串列表，能够很好地在 agate 库中使用。

标题已经准备完毕，现在需要从 Excel 文件中选择要使用的数据行。我们的表单有国家和大洲的数据，让我们首先聚焦于国家的数据。我们想要避免意外地将不同类别的数据混合在一起。通过之前的代码输出，我们知道第 6 行至第 114 行是我们想要使用的。我们会使用 row_values 方法来返回 xlrd 表单对象中这些行中的值。

```
country_rows = [sheet.row_values(r) for r in range(6, 114)]
```

现在我们有了标题列表和数据列表，所以只需要定义导入 agate 库中的类型。根据关于定义列的文档（http://agate.readthedocs.org/en/latest/tutorial.html#defining-the-columns），我们有文本、布尔、数字和日期 4 种列类型，同时库作者建议，如果数据的类型不确定，就使用文本类型。这里同样有内置的 TypeTester（http://agate.readthedocs.io/en/latest/api/data_types.html），可以用其猜测数据类型。首先，使用一些 xlrd 的内置函数来定义列：

```
from xlrd.sheet import ctype_text
import agate

text_type = agate.Text()
number_type = agate.Number()
boolean_type = agate.Boolean()
date_type = agate.Date()

example_row = sheet.row(6)

print example_row ❶
```

```
print example_row[0].ctype ❷
print example_row[0].value

print ctype_text ❸
```

❶ 通过打印来检查这一行的值，我们看到有完好的数据。xlrd 会检查所有的数据，保证不会有空行数据的存在。

❷ 这两行代码中，我们调用 ctype 和 value 属性来得到数据行中每一个元素的类型和值属性。

> 当使用 IPython 时，通过创建一个你关心的变量的新对象，在末尾添加句号，并敲击 Tab 键，就可以轻松地找到新的方法和属性。这会展开一个属性和方法的列表，以便于你更深入地探索。

❸ 使用 xlrd 库中的 ctype_text 对象，我们可以匹配 ctype 方法返回的整数对象，映射它们到可阅读的字符串。这可以代替手动映射类型。

这几行代码让我们更好地了解了可用于定义类型的工具。ctype 方法和 ctype_text 对象可以用来排序和展示给定样例数据行中的数据类型。

> 虽然看起来我们为了用这种方式创建列表做了好多工作，但这些工作提供了可复用的能力，这会在之后节省你的时间。重用这些代码片段会在之后为你节省大量宝贵时间，同时也是编写你自己的代码中非常有趣的一面。

现在我们知道哪些函数可以用来探索 Excel 列的数据类型，所以需要尝试为 agate 库创建一个类型列表。我们需要遍历数据行，使用 ctype 来映射列类型：

```
types = []

for v in example_row:
    value_type = ctype_text[v.ctype] ❶
    if value_type == 'text': ❷
        types.append(text_type)
    elif value_type == 'number':
        types.append(number_type)
    elif value_type == 'xldate':
        types.append(date_type)
    else:
        types.append(text_type) ❸
```

❶ 映射我们在探索每一行数据的 ctype 属性时找到的整数值到 ctype_text 字典中，使它们变得可读。现在 value_type 中保存了数据的列类型字符串（也就是文本、数字等）。

❷ 使用 if 和 elif 语句以及 == 操作符将 value_type 和 agate 列类型匹配。之后，代码将相应的类型追加到列表中，继续下一个列类型。

❸ 正如库文档中建议的那样，如果这里没有类型匹配上，我们将文本列类型追加到列表中。

现在我们构建了一个函数来接受一个空列表，遍历所有的列，并且创建一个包含数据集中所有列类型的列表。在运行代码之后，我们就有了需要的类型、标题和数据列表。可以将标题和类型打包在一起，通过运行下面这行代码，将结果导入到 agate 表中：

```
table = agate.Table(country_rows, titles, types)
```

当你运行这段代码时，会看到 CastError，同时还有 Can not convert value “-” to Decimal for NumberColumn 这行错误信息。

正像第 7 章和第 8 章提到的那样，学习如何清洗数据是数据处理过程中必要的一部分。编写文档完备的代码会让你在未来节省时间。通过阅读这些错误信息，我们意识到有一些坏数据隐藏在某个数据列中。在表单中的一些地方，数据中出现了 '-'，而不是 ''，这会被当作空值来处理。我们可以编写一个函数来处理这个问题。

```
def remove_bad_chars(val): ❶
    if val == '-': ❷
        return None ❸
    return val

cleaned_rows = []

for row in country_rows:
    cleaned_row = [remove_bad_chars(rv) for rv in row] ❹
    cleaned_rows.append(cleaned_row) ❺
```

❶ 定义函数来去除坏字符（例如整数列中的 '-' 字符）。
❷ 如果值与 '-' 相等，选择这个值准备替换。
❸ 如果值为 '-'，返回 None。
❹ 遍历 country_rows 来创建一个新的清洗后的列表，包含合法的数据。
❺ 创建一个 cleaned_rows 列表包含清洗后的数据（通过 append 方法）。

> 当我们编写函数来修改值的时候，在主逻辑之外保证一个默认的返回（像示例中那样），以确保永远返回一个值。

使用这个函数，我们可以确保整数列拥有 None 类型，而不是 '-'。None 告诉 Python，这是一个空数据，在与其他数字比较分析时忽略这个数据。

因为想要复用清洗和改变类型的代码，所以我们把一些已经编好的代码转变成更加抽象和通用的辅助函数。创建最后的清洗函数时，我们创建了一个新的列表，遍历所有行的数据，对每一行数据做了清洗，并为 agate 表返回了一个新的数据列表。让我们看一下，是否可以使用这些概念并抽象它们。

```
def get_new_array(old_array, function_to_clean): ❶
    new_arr = []
    for row in old_array:
        cleaned_row = [function_to_clean(rv) for rv in row]
        new_arr.append(cleaned_row)
    return new_arr ❷

cleaned_rows = get_new_array(country_rows, remove_bad_chars) ❸
```

❶ 定义函数，让其接受两个参数：老的数据数组和清洗数据的函数。
❷ 用更抽象的名称复用我们的代码。在函数的最后，返回新的清洗后的数组。

❸ 使用 remove_bad_chars 函数作为参数调用这个函数，保存清洗后的结果到 cleaned_rows。

现在尝试重新运行代码来创建一个表：

```
In [10]: table = agate.Table(cleaned_rows, titles, types)

In [11]: table
Out[11]: <agate.table.Table at 0x7f9adc489990>
```

呜啊！我们有了一个 table 变量，保存着一个 Table 对象。现在可以用 agate 库的函数查看数据了。如果你急切地想知道表看起来是什么样的，使用 print_table 方法快速地查看一下表中的内容：

```
table.print_table(max_columns=7)
```

> 如果至此你一直使用 IPython，并且想确保可以在下一个会话中继续使用这些变量，就使用 %store（https://ipython.org/ipython-doc/3/config/extensions/storemagic.html）命令。如果希望保存 table 变量，我们可以简单地输入 %store table。在我们的下一个 IPython 会话中，可以通过输入 %store -r 恢复 table 变量。在分析数据的过程中，这对“保存”你的工作非常有用。

下面，我们会深入查看表数据，并使用一些内置的研究工具。

9.1.2 探索表函数

agate 库提供了许多函数来探索数据。首先，我们尝试一些排序方法（http://agate.readthedocs.org/en/latest/tutorial.html?#sorting-and-slicing）。让我们尝试为表排序。通过对童工雇用率的总百分比的列排序，我们可以看到最过分的国家。我们会使用 limit（http://agate.readthedocs.io/en/latest/api/table.html）函数来查看雇用率最高的 10 个国家。

```
table.column_names ❶

most_egregious = table.order_by('Total (%)', reverse=True).limit(10) ❷

for r in most_egregious.rows: ❸
    print r
```

❶ 检查列名称，这样我们知道要使用的是什么列。

❷ 链式调用 order_by 和 limit 方法来创建新的表。因为 order_by 会按从最小到最大来排序，所以我们使用 reverse 参数来让其从大到小排序。

❸ 使用新表的 rows 属性，遍历童工雇用情况最糟糕的 10 个国家。

运行这段代码会返回童工雇用率最高的 10 个国家。在儿童工作百分比方面，非洲国家大量出现在列表前列。这是我们第一个有趣的发现！让我们继续探索。为了探究哪些国家的女童雇用率最高，我们可以再一次使用 order_by 和 limit 函数。这一次，我们将它们应用到女童百分比数据列上：

```
most_females = table.order_by('Female', reverse=True).limit(10)
for r in most_females.rows:
```

```
    print '{}: {}%'.format(r['Countries and areas'], r['Female'])
```

> 当第一次探索数据时，使用 Python 的 format 函数会让输出更易阅读，而不是简单地输出每一行数据。这意味着你可以专注于数据本身，而不是费劲去阅读它们。

我们看到了数据中有一些 None 的百分比数据。这不是我们想要的！我们可以使用 agate 表的 where 方法清除这些数据，像下面的代码一样。这一方法类似于 SQL 中的 WHERE 语句，或者 Python 中的 if 语句。where 创建另外一个只包含符合条件的数据行的表。

```
female_data = table.where(lambda r: r['Female'] is not None)
most_females = female_data.order_by('Female', reverse=True).limit(10)

for r in most_females.rows:
    print '{}: {}%'.format(r['Countries and areas'], r['Female'])
```

首先创建 female_data 表，其中使用了 Python 的 lambda 函数来保证每一行数据有 Female 列存在。where 函数接受 lambda 函数返回的布尔值，并且只在值为真时将数据分离出来。将只含有女性童工雇用数据的行分离出来后，我们使用相同的排序、截断和格式化技巧，来查看女童雇用率非常高的国家列表。

lambda

Python 的 lambda 函数允许我们编写一个单行函数，并且作为一个参数进行传递。这对于我们在这一节探索中所碰到的情况来说十分有用，在这里，我们希望通过一个简单的函数传递一个值。

在编写 lambda 函数时，像上面例子中那样，我们首先编写 lambda 和我们会传递到函数中的代表参数的变量。在这个例子中，变量是 r。在变量名称之后，我们编写一个冒号（:）。这和我们用 def 定义函数，并且用冒号终结一行是相同的。

在冒号之后给 Python 我们想要 lambda 函数计算的逻辑，这个函数会返回一个值。在这个例子中，返回一个布尔值，告诉我们每行中 Female 这个值是否非 None。不一定要返回布尔值，lambda 可以返回任何类型的值（整型、字符串、列表，等等）。

同样可以在 lambda 函数中使用 if else 语句，基于简单的逻辑返回一个值。在 Python 解释器中尝试下面的代码：

```
(lambda x: 'Positive' if x >= 1 else 'Zero or Negative')(0)  ❶
(lambda x: 'Positive' if x >= 1 else 'Zero or Negative')(4)
```

❶ 将 lambda 函数放在第一对括号中，将要使用的变量作为 x 放在第二对括号中。这个 lambda 函数检查参数是否等于或大于 1。如果是，返回 Positive，否则返回 Zero or Negative。

lambda 函数极为有用，但是也会使代码更加难以阅读。要确保遵守好的编程规则，并且只在清晰明确的情形下使用它们。

在检视数据的时候，我们发现很多国家既拥有高童工雇用率又拥有高女童雇用率。我们看过一些过滤和排序的方法，再来看一些 agate 库内置的统计学函数。假如我们想要找到城市童工雇用率的平均百分比。为此，我们来计算 Place of residence (%) Urban 这列数据的均值：

```
table.aggregate(agate.Mean('Place of residence (%) Urban'))
```

这段代码调用了表的 aggregate 方法，使用 agate.Mean() 统计学方法和列名称来返回列的数学均值。你可以通过 agate 文档（http://agate.readthedocs.io/en/latest/cookbook/statistics.html#statistics）来查看其他可以在列上使用的聚合函数。

当运行这段代码时，会收到 NullComputationWarning 异常。从这个异常的名字和过往经验中，你可能猜到了这个异常的意义，这个异常意味着 Place of residence (%) Urban 列中可能有一些空数据。我们可以再次使用 where 方法来聚焦于城市平均值：

```
has_por = table.where(lambda r: r['Place of residence (%) Urban'] is not None)

has_por.aggregate(agate.Mean('Place of residence (%) Urban'))
```

你会发现得到了相同的值，这是因为 agate 在背后做了相同的事情（去除空列，计算剩下数据的平均值）。让我们来看看对于居住信息表还可以做什么数学计算。可以看一下 place of residence 列的最小值（Min）、最大值（Max）和均值（Mean）。

假如我们想要找到每行数据中农村童工雇用率大于 50% 的数据。agate 库有一个 find 方法，使用条件语句来找到第一个匹配的数据。让我们尝试用代码解决问题：

```
first_match = has_por.find(lambda x: x['Rural'] > 50)

first_match['Countries and areas']
```

返回的那行数据就是第一个匹配到的数据，就像在普通字典中一样，我们可以看到数据的名称。在 agate 库的第一次探索之旅中，最后一步，我们将会使用 compute 方法和 agate.Rand() 统计学方法（http://agate.readthedocs.io/en/latest/cookbook/rank.html），基于另一列创建一个新的排序的列。

当比较数据集的时候，基于一列数据对整体数据进行排序是一个很好的彻查方式。

为了查看童工雇用率最高国家的排名，我们可以使用 Total (%) 列数据进行排序。在将这个数据集和其他数据集合并之前，我们想要一个清晰可见的排序后列数据，来比较合并后的数据。因为我们想要雇用率更高的国家出现在列表前面，所以需要使用参数 reverse=True 逆序排序（http://agate.readthedocs.io/en/latest/cookbook/rank.html#rank-descending）。

```
ranked = table.compute([('Total Child Labor Rank',
                         agate.Rank('Total (%)', reverse=True)), ])

for row in ranked.order_by('Total (%)', reverse=True).limit(20).rows:
    print row['Total (%)'], row['Total Child Labor Rank']
```

如果想用另一种方式来计算排名，可以用逆百分比创建一列数据。相对于使用每个国家

雇用童工的百分比数据，我们可以使用普通儿童的占比来进行计算。这会让我们在使用 agate.Rank() 方法时，不需要 reverse 参数：

```
def reverse_percent(row): ❶
    return 100 - row['Total (%)']

ranked = table.compute([('Children not working (%)',
                         agate.Formula(number_type, reverse_percent)),
                        ]) ❷

ranked = ranked.compute([('Total Child Labor Rank',
                          agate.Rank('Children not working (%)')),
                        ]) ❸

for row in ranked.order_by('Total (%)', reverse=True).limit(20).rows:
    print row['Total (%)'], row['Total Child Labor Rank']
```

❶ 创建一个新的函数来计算并返回给定数据的逆百分比。

❷ 使用 agate 库的 compute 方法，传递一个列表作为参数，并返回新的数据列。列表中的每一个元素必须是元组对象，而元组的第一个元素包含列名称，第二个元素用来计算新的列。在这里，我们使用 Formula 类，其同样需要一个 agate 类型，同函数一起，创建一个列表值。

❸ 用 Children not working (%) 列的数据来创建有适当排序的 Total Child Labor Rank 列。

可以看到，compute 是一个非常好用的工具，它基于一个数据列（或多个数据列）来计算一个新的数据列。现在我们有了排名，让我们看看是否能够合并一些新数据集到童工数据集里。

9.1.3　联结多个数据集

在研究可以和童工数据联结的数据集时，我们碰到了很多无果而终的情况。我们尝试使用世界银行数据（http://data.worldbank.org/）来比较农业和服务经济数据，但是并没有找到任何好的联系。我们进行了更多的阅读，发现有些人把童工和 HIV 感染率联系了起来。我们观察了这些数据，但是并没有找到明显的总体趋势。顺着这个思路，我们想知道犯罪比率对童工雇用率是否有影响——但是这一次，我们依然没有发现任何关联。[1]

经历这么多失败之后，在仔细考察数据和阅读一些文章时，一个特别的想法突然出现。政府腐败（或政府被认为有可能存在腐败）会不会影响童工雇用率？当我们阅读关于童工雇用的资料时，经常发现与反政府武装、学校和工业相关。 如果大众不相信政府，并且必须创建一些未经政府批准的组织，这些都可能是招募这些希望工作和帮忙的人（甚至是儿童）的原因之一。

我们锁定了国际公开腐败感指数（Transparency International's Corruption Perceptions Index）数据集，并决定与 UNICEF 童工数据做比对。首先，我们需要把数据导入到 Python 中。下面代码交代了如何导入数据：

```
cpi_workbook = xlrd.open_workbook('corruption_perception_index.xls')
```

注 1：查看本书的仓库，可以看到其中的一些探索。

```
cpi_sheet = cpi_workbook.sheets()[0]

for r in range(cpi_sheet.nrows):
    print r, cpi_sheet.row_values(r)

cpi_title_rows = zip(cpi_sheet.row_values(1), cpi_sheet.row_values(2))
cpi_titles = [t[0] + ' ' + t[1] for t in cpi_title_rows]
cpi_titles = [t.strip() for t in cpi_titles]

cpi_rows = [cpi_sheet.row_values(r) for r in range(3, cpi_sheet.nrows)]

cpi_types = get_types(cpi_sheet.row(3))
```

我们再一次使用 xlrd 来导入 Excel 数据，并且复用在之前编写的解析标题和为 agate 库准备数据的代码。但是在你运行最后一行代码（这里调用了一个新的函数，get_types）之前，我们需要编写一些代码来帮助我们定义类型和创建表：

```
def get_types(example_row):
    types = []
    for v in example_row:
        value_type = ctype_text[v.ctype]
        if value_type == 'text':
            types.append(text_type)
        elif value_type == 'number':
            types.append(number_type)
        elif value_type == 'xldate':
            types.append(date_type)
        else:
            types.append(text_type)
    return types


def get_table(new_arr, types, titles):
    try:
        table = agate.Table(new_arr, titles, types)
        return table
    except Exception as e:
        print e
```

我们使用之前编写的相同代码来创建函数 get_types，它接受一行数据，为 agate 库输出一个类型列表。我们同样编写了 get_table 函数，函数中使用了 Python 内置的异常处理。

异常处理

整本书中，我们碰到了很多错误，并在它们发生时进行了处理。现在我们有了更多经验，可以开始预见潜在的错误，并做出适当的决定来处理它们。

代码要明确（特别是异常），这样就能够在代码中说明你预期的错误。这还会确保没有预见到的错误会抛出异常，进入错误处理的逻辑，输出错误日志，停止程序的执行。

当使用 try 和 except 时，我们告诉 Python："请尝试执行这段程序。如果你碰到了错误，请停止执行前一节代码，执行 except 代码块中的代码。"下面是一个例子：

```
try:
    1 / 0
except Exception:
    print 'oops!'
```

这个例子使用的是通用的异常类型。通常我们希望捕获特定类型的异常，即认为程序会抛出的异常。例如，如果代码会将字符串转化为数字，我们就知道可能会出现一个 ValueError 异常。可以像下面这段代码一样处理这个情况：

```
def str_to_int(x):
    try: ❶
        return int(x) ❷
    except ValueError: ❸
        print 'Could not convert: %s' % x ❹
    return x
```

❶ 开始 try 代码块，它定义了代码可能抛出异常。try 关键字后面永远跟着一个冒号，并且单独占据一行空间。下面一行或者几行的代码，是一个 Python 的 try 代码块，使用 4 个空格缩进。

❷ 返回传入参数的整数形式。当参数是类似于 1 或者 4.5 的值时，这不会有问题。如果参数的值是 '-' 或者 'foo'，这会抛出一个 ValueError 异常。

❸ 开始 except 代码块，定义需要捕获的异常类型。这一行同样使用一个冒号作为结束，指定我们想要捕获一个 ValueError 异常（这样 except 代码块会只捕获 ValueError 异常）。这个代码块和下面的代码只在 try 语句抛出了这行代码中指定的异常时才会执行。

❹ 打印一行信息，告诉关于异常的信息。如果需要更新或改进代码，我们可以使用这段信息。

一般来说，我们想构建简明且明确的 try 和 except 代码块。这会让代码变得易读、可预测又明确。

你可能会问，为什么在之后编写的 get_table 函数中使用了 except Exception？这是个好问题！我们总是希望明确代码；然而，当你第一次用一个库或数据集进行实验的时候，可能不清楚该预防哪些错误。

为了编写捕获特定异常的代码，你需要预测代码可能会抛出什么类型的异常。有 Python 内置的异常类型，但是也有你不熟悉的特殊库异常。如果你正在使用一个 API 库，作者可能会编写一个 RateExceededException 异常，来告诉你发送了太多的请求。当我们面对一个全新的库时，使用 except Exception 代码块，打印或记录日志会帮助我们更多地了解这些错误。

> 当编写 except 代码块时，可以通过在异常后面添加代码 as e（在冒号前）存储异常到一个变量 e。因为打印了包含异常信息的变量 e，所以可以了解到更多关于触发异常的信息。最终我们会用更精确的异常，或者一系列的异常代码块，重新编写 except Exception 块，这样代码会运行得更顺利和可预测。

现在我们有了一个 get_table 函数来跟踪 agate 库的异常，并可考虑如何改进代码。我们

可以使用新函数将腐败指数数据导入到 Python 中。尝试运行下面的代码：

```
cpi_types = get_types(cpi_sheet.row(3))

cpi_table = get_table(cpi_rows, cpi_types, cpi_titles)
```

大功告成！当你运行这段代码，我们的新函数 `get_table` 会让你看到抛出的错误，而不是函数完全中断。重复的标题可能意味着 = 标题列表中有一些坏标题。通过运行下面的代码来查看这个问题：

```
print pci_titles
```

可以看到有两个 Country Rank 列。通过在电子表格中查看 Excel 数据，我们看见的确有重复的列。为了方便起见，我们不会考虑去除重复数据，但是的确需要处理重复的列名称。我们需要拼接 Duplicate 到其中的一个列名称。下面的代码展示了如何处理重复列名称：

```
cpi_titles[0] = cpi_titles[0] + ' Duplicate'

cpi_table = get_table(cpi_rows, cpi_types, cpi_titles)
```

我们将第一个标题替换为 Country Rank Duplicate，并且再次尝试创建新的 `pci_table`:

```
cpi_rows = get_new_array(cpi_rows, float_to_str)

cpi_table = get_table(cpi_rows, cpi_types, cpi_titles)
```

现在我们有了没有任何差错的 `cpi_table`。我们可以着手将其与童工数据联结起来，查看它们之间有什么样的联系。在 `agate` 库中，有一个很好用的联结表的方法：`join`（http://agate.readthedocs.io/en/latest/api/table.html#agate.table.Table.join）。`join` 方法模仿 SQL 中的语义，将两张表通过一个共享键联结到一起。表 9-1 总结了不同的联结方式和对应的功能。

表9-1：表联结

联结方式	功能
左外联结	保留左侧表（或 `join` 语句中的第一张表）中的所有行数据，使用共享键来绑定右侧表（或 `join` 中的第二张表）的值。如果右侧表中没有匹配的值，使用空值来填充
右外联结	使用右侧表中的值作为初始的匹配键。如果在第一张表（左侧表）中没有对应的值，这些行会使用空值填充
内联结	只返回两张表中都能够通过共享键匹配到值的数据行
全外联结	保留两张表中的所有数据，当共享键相匹配时，组合两张表中的数据到一行中

如果数据并不是完全匹配，或者不具备一对一关系，同时你在使用外联结，就会有为空值的数据行。当表不匹配时，外联结保持表中的数据存在，并用空值替代缺失的数据。这在因为报告必需而想要保留不匹配的数据时非常有用。

如果想要联结 `table_a` 和 `table_b`，同时确保不会丢失任何 `table_a` 里的数据，那可以编写下面的代码：

```
joined_table = table_a.join(
    table_b, 'table_a_column_name', 'table_b_column_name')
```

在结果 `joined_table` 中，我们会得到基于传递的列名称，和 `table_a` 匹配的所有 `table_b` 值。如果 `table_a` 中有值不匹配 `table_b`，我们会保留这些行，但是它们会在 `table_b` 列上

成为空值。如果 table_b 中有值没有在 table_a 中匹配，它们会被排除在新表之外。选择哪一张表在前面和使用哪一种方式的联结是非常重要的。

我们绝不想要空值存在。我们的问题围绕数据是怎样关联的，为了达到这个目的，我们想要使用内联结。agate 库的 join 方法允许传递 inner=True 参数，这会使函数仅作内联结，只保留匹配的行，不会在联结后有空值行。

我们尝试联结童工数据和新规整后的 cpi_table。当我们查看这两个表时，可以将它们通过国家 / 领土的名称匹配在一起。在 cpi_table 中，我们有 Country/Territory 列，同时，在童工数据中，我们有 Counties and areas 列。为了联结这两张表，运行下面的代码：

```
cpi_and_cl = cpi_table.join(ranked, 'Country / Territory',
                            'Countries and areas', inner=True)
```

将匹配行放到新表 cpi_and_cl 中。我们可以通过打印几个值来查看这张表，同时研究新的联结后的列，像下面代码这样：

```
cpi_and_cl.column_names

for r in cpi_and_cl.order_by('CPI 2013 Score').limit(10).rows:
    print '{}: {} - {}%'.format(r['Country / Territory'],
                                r['CPI 2013 Score'], r['Total (%)'])
```

当查看列名称时，可以看到现在有了两张表里面的所有的列。对数据进行简单计数返回 93 行。我们不需要所有的数据点（pci_table 有 177 行，ranked 有 108 行），我们想要看到的就是关联在一起的数据。当使用 CPI 得分排序，打印出新的联结的表时，是否注意到一些其他的东西？我们只选择了最高的 10 行数据，但是一些有意思的信息变得清晰：

```
Afghanistan: 8.0 - 10.3%
Somalia: 8.0 - 49.0%
Iraq: 16.0 - 4.7%
Yemen: 18.0 - 22.7%
Chad: 19.0 - 26.1%
Equatorial Guinea: 19.0 - 27.8%
Guinea-Bissau: 19.0 - 38.0%
Haiti: 19.0 - 24.4%
Cambodia: 20.0 - 18.3%
Burundi: 21.0 - 26.3%
```

除了伊拉克（Iraq）和阿富汗（Afghanistan）两个国家，当国家有非常低的 CPI 得分（即政府腐败高概率）时，同时有非常高的童工雇用率。使用 agate 库的一些内置方法，我们可以研究数据集中存在的类似关系。

9.1.4 识别相关性

agate 库有一些很好的工具，供你在数据集上做简单的数据分析。这是一个很好的初始工具集——可以从 agate 库的工具开始，之后转向更高级的数据分析库，包括 pandas、numpy 和 scipy，按需选择。

我们想要确定政府腐败和童工雇用率之间是否有关联。我们将使用的第一个工具是简单的皮尔森相关系数（http://onlinestatbook.com/2/describing_bivariate_data/pearson.html）。agate

库基于这个算法开发了 agate-stat 库（https://github.com/onyxfish/agate-stats）。在这之前，你可以使用 numpy 进行相关性分析。相关系数（例如皮尔森相关系数）告诉我们数据是否关联，以及一个因子是否会影响另一个因子。

如果你还没有安装 numpy，可以通过运行命令 pip install numpy 安装它。之后，使用下面几行代码计算童工雇用率和政府腐败指数之间的相关性：

```
import numpy

numpy.corrcoef(cpi_and_cl.columns['Total (%)'].values(),
               cpi_and_cl.columns['CPI 2013 Score'].values())[0, 1]
```

我们首先得到了类似于之前曾见到的 CastError 异常的错误。因为 numpy 需要浮点型数据，而不是小数型，所以我们需要将数字转换回浮点数。我们可以使用列表生成式做这个转换：

```
numpy.corrcoef(
    [float(t) for t in cpi_and_cl.columns['Total (%)'].values()],
    [float(s) for s in cpi_and_cl.columns['CPI 2013 Score'].values()])[0, 1]
```

我们的输出显示出一些轻微的负相关：

```
-0.36024907120356736
```

负相关意味着，一个变量增长，另一个变量会减小。正相关意味着两个变量会同时增长或减小。皮尔森相关系数在 −1 到 1 之间波动，0 意味着无相关性，−1 和 1 意味着相关性很强。

我们的结果是 −0.36，意味着弱相关，但是的确有关联。我们可以使用这个结果更加深入地研究这一数据集，搞清楚其中的含义。

9.1.5 找出离群值

随着数据分析的进行，你会想要使用一些其他的统计学方法来解释你的数据。一个入手点就是找出离群值。

离群值出现在个别的数据明显有别于数据集其他部分的时候。离群值会告诉我们数据的一部分情况。有时候，去掉它们会展现出一个明显的趋势；有些时候，离群值本身会透露出很多信息。

有了 agate 库，找到离群值是很容易的。有两种方法可以做到这一点：一是使用标准差，第二个是使用绝对中位差。如果你有一些统计学知识，并且想要使用其中的一个，尽情去尝试！如果没有相关的统计学基础，在你的数据集上同时使用两种方式来分析偏差，可能会揭露出不同的结果。[2]

注 2：更多关于绝对中位差和标准差的信息，查看 Matthew Martin 关于为什么我们仍在使用标准差的文章（http://www.separatinghyperplanes.com/2014/04/why-do-statisticians-use-standard.html），以及 Stephen Gorad 关于为什么及何时使用均差的学术文章（http://www.leeds.ac.uk/educol/documents/00003759.htm）。

如果你已经知道数据的分布，可以采用适当的方式来确定变化值；但是在你第一次探索数据时，尝试使用多种不同的方法来确定数据的分布，并了解数据的组成。

我们将会使用 agate 表的标准差离群值（http://agate.readthedocs.io/en/latest/cookbook/statistics.html#identify-outliers）方法。这个方法返回包含至少 3 个高于或低于平均值的偏差值表。下面是使用 agate 表查看标准差离群值的方法。

如果你在使用 IPython 处理数据，并且需要安装新的库，可以使用 IPython 的魔法命令 `%autoreload`，在另外一个终端安装新的库后重新加载你的 Python 环境。尝试执行 `%load_ext autoreload`，然后执行 `%autoreload`。没错！你拥有了新的库，却没有丢失任何的进程信息。

首先，需要通过运行命令 `pip install agate-stat` 安装 `agate-stat` 库。然后运行下面的代码：

```
import agatestats
agatestats.patch()

std_dev_outliers = cpi_and_cl.stdev_outliers(
    'Total (%)', deviations=3, reject=False) ❶

len(std_dev_outliers.rows) ❷

std_dev_outliers = cpi_and_cl.stdev_outliers(
    'Total (%)', deviations=5, reject=False) ❸

len(std_dev_outliers.rows)
```

❶ 使用童工雇用数据的 Total (%) 列和 `agate-stats stdev_outliers` 方法来查看我们的数据中是否含有容易找到的标准差离群值。我们将这个方法的输出赋值给一个新的表 `std_dev_outliers`。我们使用参数 `reject=false` 来告诉函数我们希望看到离群值。如果我们设置 `reject` 等于 `True`，将会得到没有离群值的数据。

❷ 查看我们发现了多少行离群值（这张表共有 94 行数据）。

❸ 提高偏差的大小，减少离群值的数量。（`deviations=5`。）

从输出中看出，我们对数据分布并没有很好的了解。当我们使用 Total (%) 列，尝试使用 3 作为标准差识别离群值时，得到了和当前表完全匹配的一张表。这不是我们想要的结果。当使用 5 作为差值界限时，我们并没有看到数据的变化。这告诉我们数据并不是常规的分布。为了找到数据中真正的偏差，我们需要更深入地探索来确定是否需要重新划分数据，使其变为我们研究的国家的子集。

可以使用平均绝对偏差检查 Total (%) 列数据的偏差：

```
mad = cpi_and_cl.mad_outliers('Total (%)')

for r in mad.rows:
    print r['Country / Territory'], r['Total (%)']
```

有趣！我们的确找到了一个更小的离群值的子集，但是得到了一个奇怪的结果列表：

```
Mongolia 10.4
India 11.8
Philippines 11.1
```

查看这个列表时，我们并没有看到数据中的任何最高值或最低值。这意味着，对于识别离群值来说，数据集并没有遵从正常的统计学规则。

> 取决于数据集和数据的分布，这两个方法经常会有效地展示出数据的信息。如果没有的话，就像我们这个数据集，继续搞清楚数据能够告诉我们什么联系和趋势。

在探索了数据的分布和数据分布所展现的趋势后，你会想要探索数据中的分组关系。下面这一节解释了怎样对数据分组。

9.1.6 创建分组

为了进一步研究数据，我们将要创建分组，研究分组之间的关系。agate 库提供了很多不同的工具来创建分组，还有其他一些方法来聚合这些分组，确定分组之间的联系。早些时候，我们的童工数据集中有完好的各大洲数据。让我们尝试从地理角度，按照大洲分组数据，看一下这样是否会揭露一些与政府腐败数据之间的关系或总结出其他结论。

首先，我们要解决怎样拿到大洲数据的问题。在本书的 git 仓库中（https://github.com/jackiekazil/data-wrangling），我们提供了一个 .json 文件，其中列举了不同大洲包含的国家。使用这个数据，我们可以添加一列，展示每个国家所属的大洲，以便通过大洲分组。下面是这一过程的代码：

```
import json

country_json = json.loads(open('earth.json', 'rb').read()) ❶

country_dict = {}

for dct in country_json:
    country_dict[dct['name']] = dct['parent'] ❷

def get_country(country_row):
    return country_dict.get(country_row['Country / Territory'].lower()) ❸

cpi_and_cl = cpi_and_cl.compute([('continent',
                                  agate.Formula(text_type, get_country)),
                                ]) ❹
```

❶ 使用 json 库来加载 .json 文件。如果你观察这个文件的话，会看到文件中保存了类型为字典的列表。

❷ 遍历 country_dict，将 country 作为键、continent 作为值填充到字典中。

❸ 创建函数：接受国家作为参数，返回它归属的大洲。这个函数使用了 Python 的字符串方法 lower，将大写字母替换为小写形式。.json 文件包含的都是小写的国家名称。

❹ 使用 `get_country` 函数创建一个新的列，continent。沿用相同的表名称。

现在我们有了大洲和国家数据。我们需要做一个快速的检查来确保没有遗漏任何东西。为了检查，运行下面的代码：

```
for r in cpi_and_cl.rows:
    print r['Country / Territory'], r['continent']
```

呃，看起来我们落下了一些数据，因为我们可以在一些国家中看到 `None` 类型的数据：

```
Democratic Republic of the Congo None
...
Equatorial Guinea None
Guinea-Bissau None
```

最好不要丢失这些数据，让我们看一下为什么这些行数据没有匹配上。我们只想要打印出没有匹配上的行。可以使用 `agate` 来帮忙找到这些行，运行下面的代码：

```
no_continent = cpi_and_cl.where(lambda x: x['continent'] is None)

for r in no_continent.rows:
    print r['Country / Territory']
```

你的输出应该类似于下面这样：

```
Saint Lucia
Bosnia and Herzegovina
Sao Tome and Principe
Trinidad and Tobago
Philippines
Timor-Leste
Democratic Republic of the Congo
Equatorial Guinea
Guinea-Bissau
```

可以看到，没有大洲数据的国家列表很短。我们建议只修改 earth.json 数据文件，因为这会使在未来使用相同的数据文件关联相同的数据更简单。如果你使用代码来找到异常值并匹配它们，当用新数据重做时会变得复杂，每一次更新数据都需要修改代码。

为了修复 .json 文件中的匹配问题，我们需要搞清楚为什么国家没有被找到。打开 earth.json 文件，找到 `no_continent` 表中的几个国家。例如：

```
{
    "name": "equatorial Guinea",
    "parent": "africa"
},
....
{
    "name": "trinidad & tobago",
    "parent": "north america"
},
...
{
    "name": "democratic republic of congo",
    "parent": "africa"
},
```

正如我们在 .json 文件中看到的那样，有一些细微的差别，影响了我们顺利地找到国家归属的大洲。本书的 git 仓库中同样包含了一个名叫 earth-cleaned.json 的文件，这个文件在 earth.json 文件上做出了一些必要的修改，比如添加 `the` 到刚果（Democratic Republic of the Congo，DRC）条目中，将若干条目中的 `&` 改为“and”。我们现在可以重新运行本节最初的代码，并且使用新的文件作为 `country_json` 数据。你需要重新联结表，避免重复的列（使用之前用过的联结两张表的代码）。在重新运行这两部分的代码后，你应该不会得到不匹配的国家了。

让我们尝试分组完整的大洲数据，看一下会发现什么。下面的代码做了这件事：

```
grp_by_cont = cpi_and_cl.group_by('continent')

print grp_by_cont ❶

for cont, table in grp_by_cont.items(): ❷
    print cont, len(table.rows) ❸
```

❶ 使用 `agate` 库的 `group_by` 方法，这会返回一个字典，其中键是大洲名称，值是一个新表，包含这个大洲的值。

❷ 遍历返回的字典，看一下每张表有多少行。我们将 `items` 中的键 / 值对分别赋值给变量 `cont` 和 `table`，这样 `cont` 代表键或者大洲名称，`table` 代表对应表的值。

❸ 打印我们的数据，来检查分组。我们使用 Python 的 `len` 函数来计算每一张表中的行数。

运行这段代码，我们得到了下面的输出（注意，你的顺序可能不同）：

```
north america 12
europe 12
south america 10
africa 41
asia 19
```

我们可以看到，非洲和亚洲的数据要大于其他各大洲。这让我们很感兴趣，但是 `group_by` 方法对于聚合数据来说并不十分容易。如果想要聚合数据，创建一个汇总列的话，我们需要看一下 `agate` 库中的聚合方法。

我们注意到 `agate` 表的 `aggregate` 方法（http://agate.readthedocs.io/en/latest/cookbook/tatistics.html#aggregate-statistics），这个方法接受一个分好组的表和一系列的聚合操作（例如求和）来基于分组计算一个新的列。

在查看了 `aggregate` 方法的文档后，我们最感兴趣的是大洲童工数据和腐败感指数相对比的结果。我们想要使用一些统计学方法来把每个分组当作一个整体来看待（使用中位数和平均数），但是同样需要识别出最极端的数据（CPI 得分的最小值，童工雇用率的最大值）。这会给我们一些良好的比较结果：

```
agg = grp_by_cont.aggregate([('cl_mean', agate.Mean('Total (%)')),
                             ('cl_max', agate.Max('Total (%)')),
                             ('cpi_median', agate.Median('CPI 2013 Score')),
                             ('cpi_min', agate.Min('CPI 2013 Score'))]) ❶

agg.print_table() ❷
```

❶ 在我们分好组的表上调用 aggregate 方法，传递一个元组列表作为参数，其中包括新的列名称和 agate 的聚合方法（利用新的列命名计算新列的值）。我们想要计算童工雇用百分比的平均值和最大值，还有腐败感指数（CPI）得分的中位数和最小值。根据你的问题和数据的不同，你可以使用不同的 agate 方法。
❷ 打印新表，这样我们可以从视觉上直观比较数据。

当运行这段代码后，你会看到下面的结果：

```
|---------------+--------------------------+--------+-----------+----------|
| continent     |                  cl_mean | cl_max | cpi_median | cpi_min  |
|---------------+--------------------------+--------+-----------+----------|
| south america | 12,7100000000000000000000 |   33,5 |      36,0 |      24  |
| north america | 10,3333333333333333333333 |   25,8 |      34,5 |      19  |
| africa        | 22,3487804878048780487 |   49,0 |      30,0 |       8  |
| asia          |  9,5894736842105263157894 73 |   33,9 |      30,0 |       8  |
| europe        |  5,6250000000000000000000 |   18,4 |      42,0 |      25  |
|---------------+--------------------------+--------+-----------+----------|
```

如果想更仔细地查看数据相关的图表，可以使用 agate 表的 print_bars 方法，该方法有一个标签列（这里是 *continent*）和一个数据列（这里是 *cl_max*），在 IPython 会话中打印童工雇用数据最大值的图表。输出如下：

```
In [23]: agg.print_bars('continent', 'cl_max')

continent     cl_max
south america 33,5 ▓░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
north america 25,8 ▓░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
africa        49,0 ▓░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
asia          33,9 ▓░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
europe        18,4 ▓░░░░░░░░░░░░░░░░░░░░░░
                   +----------+---------------+-------------+------------+
                  0,0        12,5            25,0          37,5         50,0
```

现在我们的大洲数据有了几种易于比较的输出，同时这张图片展示了一些趋势。我们注意到非洲的童工雇用率均值最高，最大值也最高，亚洲和南美洲紧随其后。亚洲和南美洲相对较低的均值意味着这些区域可能存在一个或多个离群值。

我们看到腐败感指数的中位值相对差别不大，欧洲最高（政府腐败感最低）。然而，当我们看最小值（最糟糕的政府腐败感得分）时，可以看到非洲和亚洲再一次获得了最糟糕的得分。

这说明该数据集中有几个故事供我们深入探索。我们能够看到政府腐败感和童工雇用之间有联系（尽管联系很弱）。我们同样可以研究哪些国家和大洲是最糟糕的童工雇用国家和最腐败的政府。我们可以看到非洲有着非常高的童工雇用率和相对更高的政府腐败感。我们知道亚洲和南美洲中存在一两个国家，相对于他们的邻居，在童工雇用率上面引人注目。

我们的聚合探索就到这里了。我们可以继续使用已创建的表来寻找更多信息，进行更深入的探索。

9.1.7 深入探索

agate 库还有其他一些强有力的特性，另外还有一些有趣的数据分析库可以用来在你的数据上进行实验。

根据你的数据和问题的不同，你可能会发现一些特性和库比其他特性和库更有用，但是我们强烈建议你找到使用不同工具来实验的方式。同数据分析本身一样，这会加深你对 Python 和数据分析库的理解。

`agate-stat` 库有一些有趣的统计方法我们还没有探索过。你可以通过 GitHub 跟踪最新的发布和功能（https://github.com/onyxfish/agate-stats）。

除此之外，我们建议你继续探索 `numpy`。你可以使用 `numpy` 来计算百分位数值（https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.percentile.html）。你同样可以更深入地使用 `scipy`，使用 `z` 得分统计方法来确定离群值（https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mstats.zscore.html）。

如果你有时间敏感的数据，`numpy` 能够计算数据之间列与列的变化，从而探索随时间推移数据的变化（https://docs.scipy.org/doc/numpy/reference/generated/numpy.diff.html）。`agate` 同样可以计算时间相关数据中列的变化（http://agate.readthedocs.io/en/1.0.0/tutorial.html#computing-new-columns）。不要忘记在组成时间列的时候使用时间类型，因为这样做你能够在时间上做一些有趣的分析（例如随时间变化的百分比变化或一系列时间变化的映射）。

如果你想要通过更多的统计方法来探索数据，可安装 `latimes-calculate` 库（http://latimes-calculate.readthedocs.io/en/latest/index.html）。这个库有很多计算方法，同样还有一些有趣的地理数据分析工具。如果你获取了一些地理数据，这个库可以提供一些有价值的工具来帮你更好地理解、映射和分析数据。

如果你想要更深入进行数理计算和分析，我们强烈推荐 Wes McKinney 的《利用 Python 进行数据分析》一书。这本书介绍了一些更健壮的数据分析库，包括 `pandas`、`numpy` 和 `scipy` 系列的库等。

花时间用一些方法利用我们之前学到的课程来探索你的数据。现在我们会进一步分析数据，确定一些可以得出结论和分享知识的方式。

9.2　分析数据

如果你已经尝试了 `agate` 库手册（http://agate.readthedocs.org/en/latest/cookbook.html，用于研究的各种不同的方法和工具的汇总）中的更多示例，就会足够熟悉数据并能开始你的分析。

数据探索和数据分析之间有哪些不同？当分析数据时，我们提出问题并且尝试使用已有的数据回答这些问题。我们可能会对数据集进行组合和分组，以构建一个统计可用的样本。而在数据探索中，我们只是想要研究数据集的一些趋势和属性，不尝试去回答特定的问题或得出确定的结论。

在一些基本的分析后，我们可以尝试回答在数据探索中发现的一些问题：

- 为什么在非洲童工雇用的概率更高？
- 在亚洲和南美洲存在什么样的童工雇用离群值？

- 腐败感和童工雇用率有什么关系？

对于你的数据集，你会有不同的问题，但是尝试跟随我们的实例，并且找到你想要探索的趋势。任何统计学上的离群值或者聚合趋势都可以将你引向有趣的问题去研究。

对我们的数据来说，最有趣的问题是，在非洲政府腐败感和童工雇用的关系。政府腐败，或者政府腐败感，是否会影响社区保护童工不被雇用的能力？

> 根据所使用的数据集和数据探索结果，你可能会有很多感兴趣、想要探索的问题。尝试聚焦于一个具体的问题，并用你的分析来回答它。针对多个具体问题重复这一过程。专注会帮助你找到好的答案，保持你的分析明确清晰。

回答这个问题需要更多的探索和更多的数据集。我们可能希望阅读更多的文章，看一下在这个主题上有哪些研究结果。我们可能还希望访问这一领域的专家。最终，我们可能希望选择非洲的一个特定地区或一系列国家，来更好地评估童工雇用情况。下面这一小节展示了怎么做这件事。

9.2.1 分离和聚焦数据

为了之后的分析，我们首先需要分离出非洲国家的数据，更加充分地探索这一子集的数据。我们已经知道了很多使用 agate 库来过滤数据的方式，所以让我们从这里开始。下面的代码展示了怎样把非洲的数据同其他数据分离开来：

```
africa_cpi_cl = cpi_and_cl.where(lambda x: x['continent'] == 'africa') ❶

for r in africa_cpi_cl.order_by('Total (%)', reverse=True).rows:
    print "{}: {}% - {}".format(r['Country / Territory'], r['Total (%)'],
                                r['CPI 2013 Score']) ❷

import numpy
print numpy.corrcoef( ❸
    [float(t) for t in africa_cpi_cl.columns['Total (%)'].values()],
    [float(c) for c in africa_cpi_cl.columns['CPI 2013 Score'].values()])[0, 1]

africa_cpi_cl = africa_cpi_cl.compute([('Africa Child Labor Rank',
                                        agate.Rank('Total (%)', reverse=True)),
                                        ])

africa_cpi_cl = africa_cpi_cl.compute([('Africa CPI Rank',
                                        agate.Rank('CPI 2013 Score')),
                                        ]) ❹
```

❶ 使用表方法 where 来过滤出所属大洲是 africa 的行。

❷ 使用某种格式化方法打印这些行，这样我们可以查看数据并进行彻底检查。我们想要确保只有非洲国家的数据，并且能够看到总童工雇用百分比和 CPI 得分。

❸ 在分离出最感兴趣的数据后，看一下皮尔森相关系数是否变化。

❹ 添加一个新的排序列，来展示我们子集中国家之间的相互排名。

在这个数据子集中，我们计算了新的皮尔森相关系数：

```
-0.404145695171
```

我们的皮尔森相关系数下降了，表示相对于全球数据，非洲数据中童工雇用和腐败感之间有着更强的相关性。

现在让我们看一下，是否可以找出一个好故事，找到我们想要去研究的数据点。我们会计算腐败感和童工雇用率的平均值，输出有最高童工雇用率和最糟糕的腐败感指数的国家（即比均值表现更差的国家）。下面是相关的代码：

```
cl_mean = africa_cpi_cl.aggregate(agate.Mean('Total (%)'))
cpi_mean = africa_cpi_cl.aggregate(agate.Mean('CPI 2013 Score')) ❶

def highest_rates(row):
    if row['Total (%)'] > cl_mean and row['CPI 2013 Score'] < cpi_mean: ❷
        return True
    return False

highest_cpi_cl = africa_cpi_cl.where(lambda x: highest_rates(x)) ❸

for r in highest_cpi_cl.rows:
    print "{}: {}% - {}".format(r['Country / Territory'], r['Total (%)'],
                                r['CPI 2013 Score'])
```

❶ 计算我们最感兴趣的列均值：腐败得分和童工雇用率。

❷ 创建函数来识别有着高童工雇用率和低 CPI 得分（即高腐败）的国家。

❸ 函数 `highest_rates` 返回 `True` 或 `False`，来选择一行数据。这个 `lambda` 函数判断国家的童工雇用率和腐败感是否高于均值。

当运行这段代码时，我们看到了一些有趣的输出。特别是下面这些行：

```
Chad: 26.1% - 19.0
Equatorial Guinea: 27.8% - 19.0
Guinea-Bissau: 38.0% - 19.0
Somalia: 49.0% - 8.0
```

我们输出了一些与均值离得不太远的位于“中部”的数据，随之是这些有着低 CPI 得分和高童工雇用率的数据。因为我们对为什么这里会有高童工雇用率，以及腐败是怎样影响童工雇用率这些问题感兴趣，所以这些数据对我们来说最为适合。

随着研究的继续，我们想要找出这些国家正在发生着什么。是否有和这些国家年轻人和童工相关的电影或文档？是否有关于这一主题的文章和书籍？是否有我们可以联系的专家或研究者？

当更加深入地观察这些国家，我们看到一些明显的事实：贩卖儿童、性虐待、非法的宗教团体、大量的小摊贩和劳动者需求。这些事实是否与选举权被剥削有关？与民众对政府不信任有关？我们是否可以追踪这些国家的民众和他们的邻里？我们是否能够找到组织或人来缓解这些问题？

随着时间的推移，观察政治和时代变化的影响是很有趣的事情。我们可以详细考察

UNICEF 的数据，或者聚焦于一个国家，利用 UNICEF 的多指标类集调查（https://www.unicef.org/statistics/index_24302.html）数据来理解过去几十年发生的变化。

对于你自己的数据集，你需要确定未来探索时有哪些可能性。你能找到更多的数据以供探索吗？你是否可以访问一些人，或者在一段较长的时间中找到一些趋势？是否有这一主题的书籍、电影或者文章能够给你更多启发？你的分析是未来研究的开始。

9.2.2　你的数据在讲什么

现在我们已经探索和分析了数据，可以开始搞清楚数据在告诉我们什么。正如我们在第一次研究童工数据集时经历的那样，有时数据并没有什么联系，没讲述任何故事，没有任何相关性。发现这一点也没关系！

> 有时候，没找到相关性将促使我们继续研究，以找到真正存在的联系。有时候，没找到联系，这件事情本身就是一个发现。

在数据分析中，我们寻找趋势和模式。正如我们在童工雇用数据中看到的那样，大多数时候分析是更深入研究的开始。和数据讲故事一样，增加人们的声音或者其他的角度，都是找到联系和发现问题的绝佳方式。

如果你发现了一些联系，即使是很弱的联系，也可以更深入地挖掘。这些联系会通向更好的问题和更专注的研究。正如我们在童工数据中看到的那样，我们对研究越专注，就越容易看到联系。从宽泛的研究开始是很好的，但是用更加精确的视角来结束研究十分重要。

9.2.3　描述结论

当你已经分析了数据并且理解了数据中的联系，就可以开始确定你能得出什么结论。真正了解你的数据集和主题非常重要，这会对你的想法提供强有力的支持。随着你的数据分析、访谈、研究的完成，你的结论会逐渐形成，你只需要确定用什么方式来向全世界分享这些结论。

> 如果你在寻找确切结论的过程中陷入困境，在你的发现中包含开放性的问题是完全可取的。一些大的故事就是从几个简单的问题开始的。

如果你能够阐明主题，指出为了得出一个更全面的结论，需要更多的文档、研究和行动的话，这本身就是一个很重要的信息。正如我们在研究中发现的那样，很难说政府腐败引发了高的童工雇用率，但是我们可以说，这之间有很弱的相关性，并且我们想要研究和分析它们之间关联的方式——特别是在某些非洲国家。

9.2.4 将结论写成文档

当你发现一些结论和更多想研究的问题之后，应该开始将你的工作成果写成文档。作为文档和最终展示成果的一部分，你需要对使用的数据源和分析的数据数量了如指掌。在我们的问题中，我们只研究了大约 90 个数据点，但它们代表了我们想要研究的部分。

你可能会发现你关注的数据集比预期的小。只要你清楚你使用的方法和使用小数据集的原因，就不会把听众和报告引入歧途。在下一章，我们会更深入地探究如何报告发现，将我们的想法和工作写成文档，把我们的想法和世界分享。

9.3 小结

在这一章，我们使用了一些新的 Python 库和技术，探索和分析了我们的数据集。你已经能够导入数据、联结数据、分组数据，并且基于发现创造新的数据集。

现在你可以使用统计学方法来找到离群值，衡量数据之间的相关性。你可以通过分离有趣的分组，并且深入数据探索之中，确定清晰的、可回答的问题来研究。如果你曾经使用过 IPython 和 `%store` 来保存变量，在下一章，我们会用这个命令做更多的交互。

现在你应该能够：

- 使用 `agate` 库评估你的数据；
- 确定哪些事，如果有的话，在数据中是至关重要的；
- 找到数据中的入手点或一部分的数据来做深入研究，得到结论；
- 通过分析和探索数据挑战你的假设。

本章中涉及的新概念和库总结在表 9-2 中。

表9-2：新的Python编程概念和库

概念/库	功能
`agate` 库	使数据分析变得简单，能够从 CSV 数据中读取数据，创建供分析的表，运行基本的数据分析函数，在数据集上应用过滤器，洞察数据
`xlrd ctype` 和 `ctype_text` 对象	当使用 `xlrd` 分析 Excel 数据时，让你能够轻松地看到数据的类型
`isintance` 函数	检验 Python 对象的类型。如何类型匹配，结果返回一个布尔值
`lambda` 函数	Python 中的单行函数，对数据集的简单过滤或解析非常有用。注意不要书写不易阅读和理解的 `lambda` 函数。如果函数很复杂，尝试用一个小函数来代替 `lambda` 函数
联结（内联结，外联结，左联结，右联结）	允许你通过一个或多个匹配的域联结两个不同的数据集。根据联结数据方式的不同（内 / 外和左 / 右），你会得到不同的数据集。花一些时间思考什么类型的联结更符合你的需求
异常处理	使你能够使用代码预见和处理 Python 异常。明确和清楚的异常捕获永远是更好的，这样你不会捕获过度泛化的异常而漏掉 bug
`numpy coerrcoef`	使用统计学方法，例如皮尔森相关系数，来确定数据集中的两部分是否有联系

（续）

概念/库	功能
agate mad_outliers 和 stdev_outliers	使用统计学模型和工具，例如标准差或平均偏差，来确定数据集是否有特殊的离群值或不合适的值
agate group_by 和 aggregate	根据特定的属性对数据集分组，通过运行聚合分析，查看在分组间是否有明显的不同之处（或相似之处）

在下一章中，你会学习如何使用可视化和讲故事的工具来在 Web 和其他媒介上分享结论。

第10章

展示数据

你已经学习了如何分析数据，现在想要展示它。针对不同听众，演示可能会有很大的差别。我们会在这一章学习多种不同类型的演示：从简单的可以在电脑上制作的演示文件到交互式的网站。

取决于你想要展示的内容，通过图表、地图或图片进行可视化可能是你尝试讲的故事中一个重要的部分。我们会学习如何构建并运行自己的站点，分享发现。还会介绍如何分享 Jupyter notebook，其他人通过它可以看到你的代码、图表、图片和结论。

首先，我们会探索如何考虑你的听众，开始讲述你通过数据分析发现的故事。

10.1 避免讲故事陷阱

讲故事并不简单。取决于主题，你可能很难从数据中得出可靠的结论。你可能会遇到不一致或不确定的数据。这没关系。建议你继续研究，也许在数据集中找到的不同示例中就蕴含着故事。

> 讲故事时所面临的一些困难是由数据分析时的个人偏见带来的。正如经济学家和记者 Allison Schranger 在“The Problem with Data Journalism”（http://qz.com/189703/the-problem-with-data-journalism/）一文中讨论的，我们不可避免地会在分析时带有偏见。她的建议是，承认这些偏见，并试着去了解数据，避免为了讲故事的目的而去曲解它。

不要擅自假定你所要讲的故事和数据是一致的。尝试先研究数据，然后讲述数据研究所得。不要花太多时间来操作数据。如果需要大量修改数据（标准化、归一化和去除离群值），你可能应该找找其他故事或不同的数据。

记住一点，讲故事是成为领域专家的重要部分。通过研究数据掌握的信息有助于你阐明新的主题和观点。理解自己的偏见，带着这样谦逊的态度，你的故事将会有效且具有启发性。

10.1.1 怎样讲故事

确定你想要讲的故事，同确定如何去讲述它一样重要。你可以使用图表、图形、时间线、地图、视频、文字和交互式的内容来讲述故事。你可以在网上发布它，或者在各种会议中演示它。你可以将它上传到一个视频分享网站。无论选择什么方式，确保讲故事的方式增强了你的发现。没有什么比看到一个糟糕的展示更令人沮丧了，它实际上抹杀了你试图讲述的故事。

在下面几个小节中，我们会评估你的听众、你的故事和可用的平台是怎样影响演示选择的。建议你阅读所有这些选择，即使你已经有了有关展示发现方式的想法。这会让你更好地理解可用的选择，即使你坚持最初的选择。对于那些面对广泛听众的演示，不同形式的组合可能是最好的选择。

确定你计划未来多久更新一次数据是讲故事的另一个部分。是持续更新吗？你的听众不久之后就能听到更多关于这个故事的信息，还是要期待年度报告？你是否能够清楚地告诉他们何时及以何种方式更新？只有你清楚听众的期望，让他们等待才是一个不错的想法。

10.1.2 了解听众

听众和内容一样重要。通过识别目标听众，你可以确定他们关于某个话题已经知道什么，什么是他们最感兴趣的内容，以及对他们来说效果最好的学习方式是什么。如果未能与听众有效沟通，那么你创建的故事就会没人感兴趣。

如果报告或展示是你工作的一部分，确定听众应该十分容易。无论是工作中的一个小组、一个执行团队，还是一个日报或年刊，你精确地了解谁会阅读你的报告。

> 如果你想向更多人展示数据，你应该研究一下已有成果，以及哪些人想要进一步深入地了解。熟悉目标领域的已有成果会帮你确定现有或潜在的听众。

如果你不确定目标听众，一个好的策略是接触对该主题明显有不同兴趣程度的不同的人，例如，一位父辈或导师、一位同事和一名学员（从你的展示和话题的角度来看）。取决于有关话题的知识水平，是否不同的人对故事中的某一部分更感兴趣？是否不同年龄和经验的听众会提出不同的问题？在讲解某个话题时，注意听众的问题，同时观察他们的反应，并基于这些观察修正关于目标听众的说明。

一旦确定了目标听众，你可以更深入地了解他们。根据听众的不同，使用下面附注栏中的建议来帮助完善讲故事的方式。

与听众交谈

当思考如何讲述故事给听众时，搞明白他们如何学习和理解这个世界，特别是你的主题，这一点很重要。这些问题会指引你讲故事的过程，将发现以最佳方式传达给目标听众。

- 你的听众是如何学习新事物的？在线学习？口口相传？通过出版物？
- 关于这个主题你的听众有多少先验知识？是否有听众会感到陌生的单词或想法？
- 你的听众能否自己探索数据？
- 你的听众会花费多少时间和注意力听故事？
- 在与你或他人谈论这个故事时，听众的参与度如何？
- 如果有新的信息发布，你的听众是否想要被告知并更新信息？

这些只是众多问题中的一部分，你可以通过这些问题来确定真正的听众，以及他们如何才能最好地消化你的故事。以这些问题为最初的提示，让它们引导你找到更多关于如何分享发现的问题。

如果你找到了听众，准备好开始讲故事了，你就可以着手研究通过可视化工具讲述数据故事的方式了。

10.2 可视化数据

处理数据时，你可能希望使用一些可视化工具来讲故事。根据故事的不同，你的可视化选择可能是图表、图片或者时间线。无论你如何展示数据，第一步是确定哪些可视化数据是有用且相关的。

在以可视化方式讲故事的过程中，确定怎样展示发现至关重要。正像 Alberto Cairo 在其关于数据可视化的博客文章（http://www.thefunctionalart.com/2014/08/to-make-visualizations-that-are.html）中写道的，如果不展示所有相关的数据，可能会让听众对你的方法和发现有疑问。

> 这类似于用文档详细描述数据分析和方法论，我们需要文档化和保护可视化探索和数据的展现，确保不会遗漏故事中重要的部分。

在这一节中，我们会探索如何使用图表、时间序列、时间线、地图、混合多媒体、文字、图片和影音来分享发现。取决于你的听众，可能会有不同类型的工具适用于你的故事。每种类型都有其优势和缺点，我们会在探索的过程中介绍这些。

10.2.1 图表

图表是分享数字数据的良好方式，尤其是比较不同的数据集或不同的分组时。如果数据有一个清晰的趋势，或者数据显示出特定的离群值，图表会帮你向听众展现这些发现。

你可以使用条形图来并排展示大量数据。比如，在《华盛顿邮报》关于婴儿死亡率的报道（http://www.washingtonpost.com/blogs/wonkblog/wp/2014/09/29/our-infant-mortality-rate-is-a-national-embarrassment/）中，Christopher Ingraham 使用了条形图来并排比较不同国家的婴儿死亡率。

为了展示随时间推移的趋势，人们通常会使用折线图。Christopher Ingraham 同样使用了折线图来比较不同年龄下的婴儿死亡率。通过条形图可以看到，美国在婴儿保护方面滞后于其他国家。折线图可以比较不同国家的婴儿死亡率随时间变化的情况，让我们从另外一个角度观察数据。

你会注意到，作者选择在折线图上仅展示少数几个国家，而不是像在条形图上展示所有的国家。他为什么做出这样的决定？可能是他检查了数据，发现包含更多国家的数据会使得图表难以阅读。

这些都是可视化发现时需要做出的各种决定。为了确定什么样的图表适合你，什么类型的图表最有用，首先定义你想要通过图表展示什么。Extreme Presentation 博客（http://extremepresentation.typepad.com/blog/2006/09/choosing_a_good.html）上有简便的流程图工具，当你开始思考这些问题时，可以从这里开始。Juice Labs 开发了一个交互式图表选择器（http://labs.juiceanalytics.com/chartchooser），展示了一些相同的概念。

> 每个图表都有其优势和弱点。如果你想要展示关系，可以使用散点图、气泡图或者折线图，所有这些都可以展示数据相关性。条形图更适合比较多个对象。如果你想要展示数据的组成或因素，可以创建一个堆叠条形图。为了展示分布，你可以使用时序图或者柱状图。

让我们考虑一下迄今为止研究过的数据，使用一些 `agate` 内置特性来绘制数据。

1. 使用`matplotlib`绘制图表

Python 的一个主要图表和图片库是 `matplotlib`，可以用来绘制数据集。它是生成简单图表的很好的方式，你越熟悉图表库，你的图片和图表会越高级。首先，需要运行 `pip install matplotlib` 来安装它。

让我们展示政府腐败感得分和童工雇用率的对比，下面是代码。

```
import matplotlib.pyplot as plt

plt.plot(africa_cpi_cl.columns['CPI 2013 Score'],
         africa_cpi_cl.columns['Total (%)']) ❶

plt.xlabel('CPI Score - 2013') ❷
plt.ylabel('Child Labor Percentage')
plt.title('CPI & Child Labor Correlation') ❸

plt.show() ❹
```

❶ 使用 `pylab` 的 `plot` 方法，传递 x 和 y 的标签数据。传递的第一个变量是 x 坐标系，第二个变量是 y 坐标系。这会创建一个 Python 图表，绘制这两个数据集。

❷ 调用 `xlabel` 和 `ylabel` 方法标记图表坐标系。
❸ 调用 `title` 方法为图表命名。
❹ 调用 `show` 方法来绘制图表。所有在调用 `show` 之前关于图表的操作，会显示在系统默认的图片程序中（例如 Preview 或 Windows 图片查看器）。标题、坐标标签和任何其他通过 `matplotlib` 设置的属性都会展示在这个图表中。

喔！ Python 渲染了图 10-1 中的图表。[1]

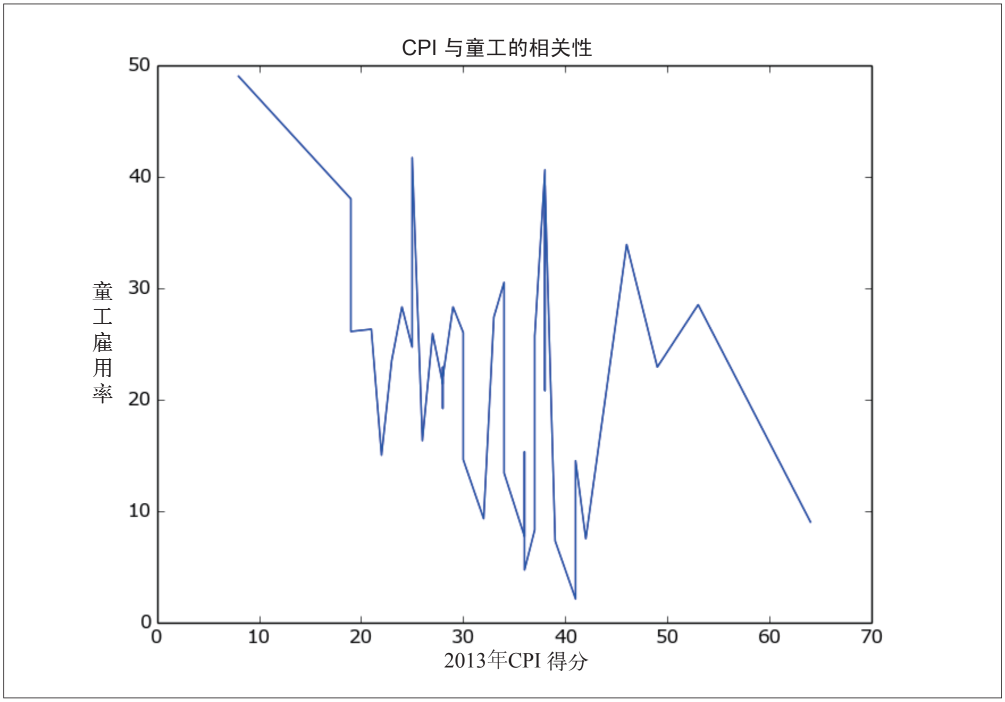


图 10-1：童工和 CPI 图表

我们能够明确看到趋势是整体下跌的，但是同样看到中部的数据并没有呈现特定的趋势。事实上，数据变化得很剧烈，这说明童工和政府腐败感并不是在所有国家都有联系，只在部分国家有联系。

让我们仅使用情况最糟糕的国家的数据来绘制图表。在 9.2.1 节中，我们已经分离出了这些国家。使用 `highest_cpi_cl` 表再次运行前面的代码，会看到图 10-2 中的图表。

现在可以看到一个清晰的下降趋势，在这些最糟糕的国家中，随着童工雇用率和政府腐败感指数的下降，出现了一些异常。

注 1：如果图表没有展示出来，根据 Stack Overflow 上的指示（http://stackoverflow.com/questions/7534453/matplotlib-does-not-show-my-drawings-although-i-call-pyplot-show/7534680#7534680），来确定 `matplotlib` 设置在哪里，并设置库的后端为默认选项之一（Mac/Linux 系统上的 QT4Agg 或 Windows 系统上的 GTKAgg）。对于 Windows 系统，可能同样需要运行 `pip install pygtk`。

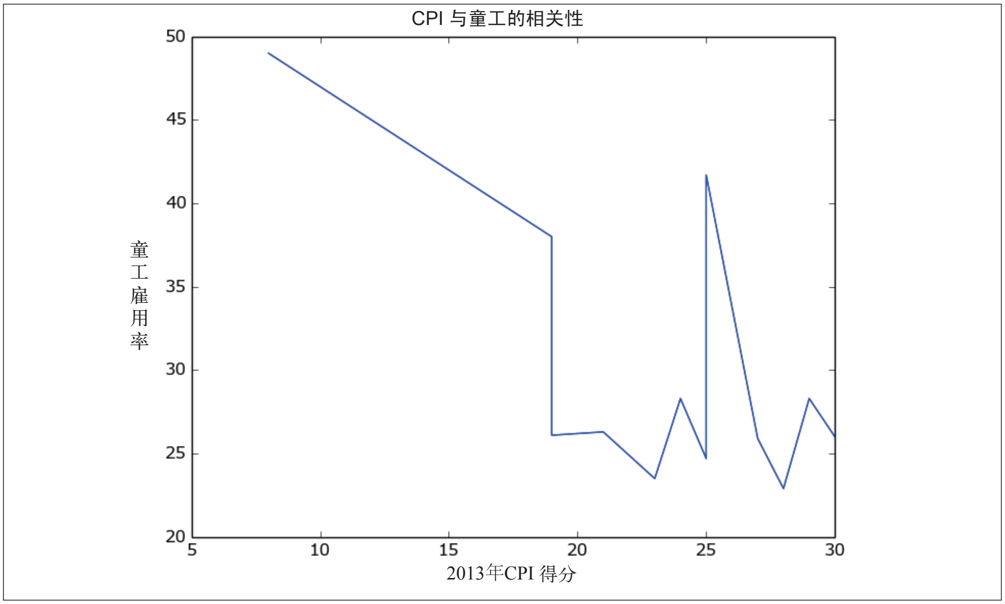
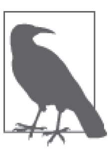


图 10-2：高童工雇用率国家图表

pylab 中有很多种图表可用，包括直方图、散点图、条形图和饼形图。强烈建议你查看一下 matplotlib.org 关于 pyplot 的介绍（http://matplotlib.org/1.4.2/users/pyplot_tutorial.html#pyplot-tutorial），包括如何改变图表的不同属性（颜色、标签、大小），使用多图、子区（subplot）和更多图表类型。



图表化数据便于发现数据集中的异常或离群值。使用 Python 图表库中各种可用的图表方法，可以帮你研究数据中的故事和关系。

越多地使用库的图表工具集，理解哪个图表最适合你的数据集就越容易。

2. 使用Bokeh绘图

Bokeh（http://bokeh.pydata.org/）是一个 Python 绘图库，能够用相当简单的命令来绘制更复杂的图表类型。如果想要创建一个条形图、散点图或时间序列图，尝试 Bokeh，看看是否合适。让我们尝试使用 Bokeh，基于各国家创建一个 CPI 和童工数据的散点图。运行下面的命令安装 Bokeh。

```
pip install bokeh
```

之后利用 agate 表使用一些简单的命令创建散点图：

```
from bokeh.plotting import figure, show, output_file


def scatter_point(chart, x, y, marker_type): ❶
    chart.scatter(x, y, marker=marker_type, line_color="#6666ee",
```

```
                        fill_color="#ee6666", fill_alpha=0.7, size=10) ❷

chart = figure(title="Perceived Corruption and Child Labor in Africa") ❸
output_file("scatter_plot.html") ❹
for row in africa_cpi_cl.rows:
    scatter_point(chart, float(row['CPI 2013 Score']),
                  float(row['Total (%)']), 'circle') ❺

show(chart) ❻
```

❶ 定义一个函数，scatter_point，接受一个图表、x 轴和 y 轴值、标记的类型（圆形、正方形、矩形），并且添加这些点到图表中。

❷ 图表的 scatter 方法需要两个必需的参数（x 轴和 y 轴）和一些不同的关键参数，为这些点添加样式（包括颜色、透明度、大小）。这行代码传递了边缘颜色和填充颜色以及大小和透明度到函数中。

❸ 使用函数 figure 创建图表，同时传入一个标题。

❹ 使用函数 output_file 定义输出的文件。这会在你运行代码的文件夹下创建文件 scatter_plot.html。

❺ 对于每一行数据，使用 CPI 得分作为 x 轴，童工雇用率作为 y 轴，添加一个数据点。

❻ 在浏览器窗口中展示这张图表。

当你运行这段代码，它会在浏览器中打开一个窗口，展示这张图表（见图 10-3）。

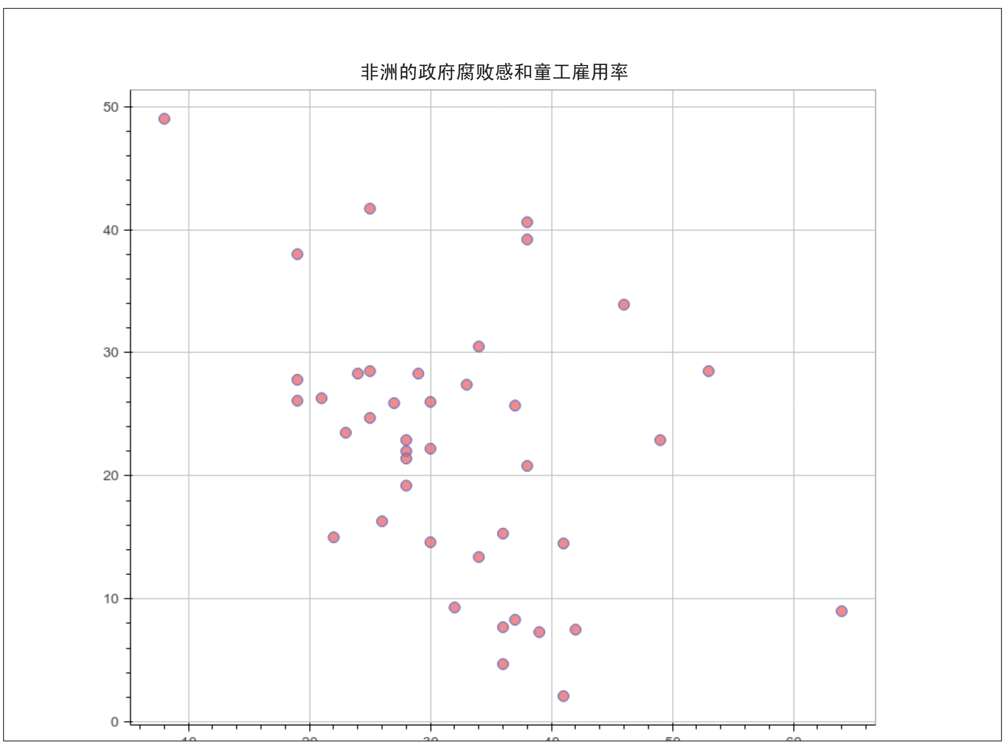


图 10-3：CPI 和童工散点图

这张图非常好，但是我们看不出这些点的含义。Bokeh 可以添加交互元素到图表中。让我们添加一些试试。

```
from bokeh.plotting import ColumnDataSource, figure, show, output_file
from bokeh.models import HoverTool ❶

TOOLS = "pan,reset,hover" ❷


def scatter_point(chart, x, y, source, marker_type): ❸
    chart.scatter(x, y, source=source,
                  marker=marker_type, line_color="#6666ee",
                  fill_color="#ee6666", fill_alpha=0.7, size=10)

chart = figure(title="Perceived Corruption and Child Labor in Africa",
               tools=TOOLS) ❹

output_file("scatter_int_plot.html")
for row in africa_cpi_cl.rows:
    column_source = ColumnDataSource(
        data={'country': [row['Country / Territory']]}) ❺
    scatter_point(chart, float(row['CPI 2013 Score']),
                  float(row['Total (%)']), column_source, 'circle')

hover = chart.select(dict(type=HoverTool)) ❻

hover.tooltips = [
    ("Country", "@country"), ❼
    ("CPI Score", "$x"),
    ("Child Labor (%)", "$y"),
]

show(chart)
```

❶ 导入我们用过的主要的库，并导入 `ColumnDataSource` 和 `HoverTool` 类。

❷ 为最终的图表定义你想要使用的工具（http://bokeh.pydata.org/en/latest/docs/user_guide/tools.html#specifying-tools）。这行代码添加了 `hover`，所以可以使用悬停方法。

❸ 把 `source` 添加到必需的参数中。这会存储国家名称信息。

❹ 传递 `TOOLS` 变量到图片初始化函数中。

❺ 变量 `column_source` 现在保存着一个数据源字典，其中是国家名称。这一行代码将国家名称作为列表传递，因为字典的值必须是一个可迭代对象。

❻ 从图表中选择 `HoverTool` 对象。

❼ 使用悬停对象的 `tooltips` 方法，展示不同的数据属性。`@country` 选择了通过列数据源传入的数据，而 `$x` 和 `$y` 选择图表中 *x* 和 *y* 数据点。

现在你的图表应该看起来类似于图 10-4。

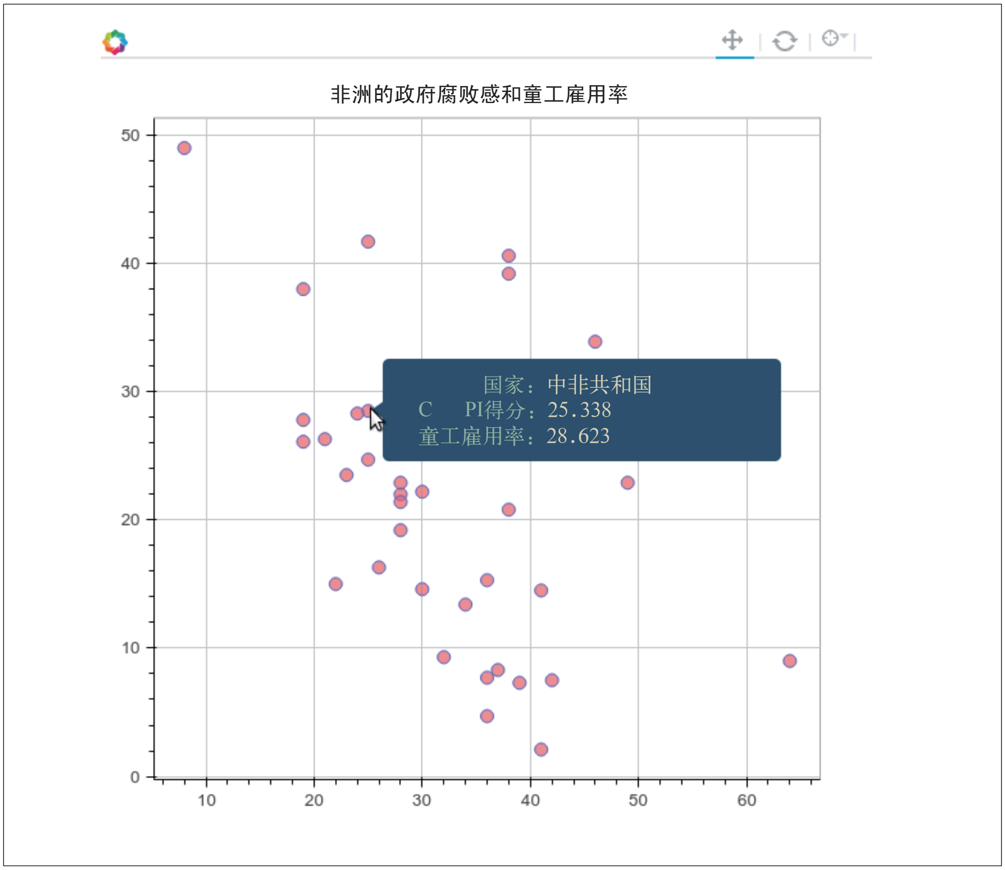


图 10-4：CPI 和童工交互散点图

> 随着你移动游标到每一个点，*x* 和 *y* 的数据会随之变化。为了优化这个图表，可以通过输入新的键值对到 `data` 字典中，添加精确的数据点到 `column_source` 对象中。

Bokeh 有一个很好的示例库（http://bokeh.pydata.org/en/latest/docs/gallery.html）和可用的代码来帮助你上手。建议你花些时间在图表上，尝试一下 Bokeh。

10.2.2 时间相关数据

时间序列和时间线数据展示随时间推移的发现。时序图表展示数据随时间的变化（通常表现为折线图、柱状图或直方图）。时间线允许你通过标记随着时间推移发生的活动、事件和变化来直观地讲述数据的故事。

1. 时间序列数据

时间序列展示随时间推移产生的趋势，尤其是当聚焦于一个因素时十分有效。《华尔街日

报》发布过一个非常棒的关于疫苗和发病率的时间序列（http://graphics.wsj.com/infectious-diseases-and-vaccines/)。交互元素允许你进行探索，内置的延时移动画创造了一个易读的画面。疫苗引进标记也增加了阅读的清晰度。

我们还没有研究数据集随时间推移的变化。下一步操作最好是收集过去几年的数据集。这些数据可以回答类似于这样的问题：随时间推移什么地方的童工数量在增长？是否能够看到随时间推移的一个明显的地区趋势？当合并另一个数据集，是否能够看到其他的趋势（例如，童工雇用数量是否伴随农业出口增长）？

在 Stack Overflow（http://stackoverflow.com/questions/19079143/how-to-plot-time-series-in-python）上有一个很好的回答，提供了更多的关于使用 `matplotlib` 来绘制时间序列数据图表的信息。还记得在第 9 章中 `agate` 表的 `rows` 和 `columns` 方法可以用来选择一列或者一行的数据吗？这两个方法返回的列表可以被传入任何 `matplotlib` 函数，把数据传入到图表中。

如果你想要了解如何使用 Bokeh 处理时间相关数据，可以查看它们很棒的实例（http://bokeh.pydata.org/en/latest/docs/user_guide/charts.html#timeseries）。

2. 时间线数据

时间线数据可以帮助你向听众介绍某主题历史中重要的时刻，或者最近发展中的转折。例如，疫苗历史网站（History of Vaccines，http://www.historyofvaccines.org/content/timelines/measles）中的时间线展示了加利福尼亚麻疹疫苗的历史和最近发展，这样听众可以通过历史数据快速地理解主题。

如果想要展示童工历史信息的时间线，我们会去查找全世界童工历史中重要的时刻。我们可以去研究帮助引出时间线上事件的问题，例如：第一部保护儿童安全的法律是什么时候实施的？什么时候公众意见开始反对童工雇用？有哪些与童工有关的公众事件和丑闻？

对于可视化数据来说，Knight 实验室的 TimelineJS（http://timeline.knightlab.com/）可以接受一个数据表单，创建一个简单的交互式时间线。

10.2.3 地图

如果你的发现聚焦于地理信息，地图是展示数据的很好的方式。地图能帮助人们意识到某主题对他们了解的人群和地区的影响。根据听众对所讨论地区的了解程度，你可能需要在地图中包含额外的信息和上下文，便于将故事与听众更熟悉的区域相关联。

如果是本地听众，你可能要提到本地知名的纪念物和街道名称。如果是国际听众，并且故事涉及一个特定的区域（例如，亚马逊森林砍伐），则首先引用大洲地图，之后再聚焦于你的目标区域。

> 地图是数据可视化中很难的一种形式。不仅你要了解听众的地理知识，而且地图并不总能用清晰易理解的方式展示趋势。当使用地图时，对展示区域的地理知识非常熟悉是很重要的，你需要展示重要的地理元素，让听众确定位置，同时展示发现。

一个有新闻价值的地图的实例是《纽约时报》的加州疫苗接种地图（http://www.nytimes.com/interactive/2015/02/06/us/california-measles-vaccines-map.html?_r=1）。该地图在加州最近的麻疹疫情爆发期间发布，读者可以放大和缩小来查看更多信息，且该地图提供了简短的描述，展示了个人意愿和其他造成低接种率的原因（例如贫穷或者缺少接触）之间的不同。通过仅仅聚焦于加利福尼亚，该地图能够展示足够详细的细节，而如果是以国家或者地区为维度的话，可能会太杂乱或者复杂了。

> 准备地图时，你可能想要利用 ColorBrewer（http://colorbrewer2.org/），这个工具可以并排比较不同的地图颜色方案。你希望颜色不仅能讲述故事，而且有对比性，这样读者可以清晰地看到组和组之间的区别。

一个更大地理区域的地图实例是《经济学人》杂志的全球债务钟（http://www.economist.com/content/global_debt_clock）。这个地图展示了各个国家的公共债务情况，使用了一个交互时间线来展示随时间推移公债的变化情况。其互补的颜色方案使得地图很容易阅读，人们可以很容易地分辨负债很重的国家和负债很少或者没有债务的国家。

> 全球债务钟地图（the global debt clock map）的作者标准化了债务度量，使用美元作为标准的展示单位，所以用户可以并排地比较不同国家的债务率。这一很小的标准化有助于听众理解并增强了这些发现的影响。

有一个很容易使用的图表和地图 Python 库，`pygal`（http://pygal.org/），它拥有很好的内置地图特性。`pygal` 有从饼状图、散点图到世界和国家地图详细的文档。可以同时使用 `pygal` 和 `agate` 表来展示世界范围内的童工雇用率。首先需要通过运行下面的命令安装库和其依赖。

```
pip install pygal
pip install pygal_maps_world
pip install cssselect
pip install cairosvg
pip install tinycss
pip install lxml
```

在 `pygal` 世界地图文档（http://www.pygal.org/en/latest/documentation/types/maps/pygal_maps_world.html?highlight=world）中，可以看到每个国家的双字符 ISO 编码是使用世界 地图所必需的。使用之前的方法添加这些编码到 `ranked` 表：

```
import json

country_codes = json.loads(open('iso-2.json', 'rb').read()) ❶
country_dict = {}

for c in country_codes:
    country_dict[c.get('name')] = c.get('alpha-2') ❷

def get_country_code(row):
    return country_dict.get(row['Countries and areas']) ❸

ranked = ranked.compute([('country_code',
```

```
                        agate.Formula(text_type, get_country_code)), ])

for r in ranked.where(lambda x: x.get('country_code') is None).rows: ❹
    print r['Countries and areas']
```

❶ 加载从 GitHub 用户 @lukes 仓库（https://github.com/lukes/ISO-3166-Countries-with-Regional-Codes）中下载的文件 iso-2.json 中的字符串。这一文件在本书仓库中也可找到。
❷ 创建国家字典，键是国家名称，值是 ISO 编码。
❸ 定义新的函数 get_country_code，接受一行数据，使用 country_dict 返回国家编码。如果没有对应键，返回 None。
❹ 查看没有匹配到的数据，做进一步的研究。

你应该看到类似下面的输出。

```
Bolivia (Plurinational State of)
Cabo Verde
Democratic Republic of the Congo
Iran (Islamic Republic of)
Republic of Moldova
State of Palestine
The former Yugoslav Republic of Macedonia
United Republic of Tanzania
Venezuela (Bolivarian Republic of)
```

我们发现大多数数据均匹配，但是有几个漏掉的。如在前一章中处理 earth.json 一样，通过修改数据文件中不匹配国家的名称手动修正这个问题。清洗后的文件，iso-2-cleaned.json，同样可以在仓库中找到。现在可以用新的、清洗后的 JSON 文件同之前的代码一起创建一个完整的表。注意，你需要重命名列或者使用新的列名称 contry_code_complete，这样不会出现重复列名称的问题。我们会利用这张表，使用 pygal 地图方法创建自己的世界地图。

```
import pygal

worldmap_chart = pygal.maps.world.World() ❶
worldmap_chart.title = 'Child Labor Worldwide'

cl_dict = {}
for r in ranked.rows:
    cl_dict[r.get('country_code_complete').lower()] = r.get('Total (%)') ❷

worldmap_chart.add('Total Child Labor (%)', cl_dict) ❸
worldmap_chart.render() ❹
```

❶ pygal 库中 map.world 模块的 World 类返回地图对象。
❷ cl_dict 保存着一个字典，键是国家编码，值是童工百分比。
❸ 根据 pygal 的文档，这行代码传递数据的标签和数据字典到函数中。
❹ 调用地图的 render 方法来展示地图。

可以看到 render 将 .svg 文件以一个很长很复杂的字符串输出到终端。如果想要保存它到一个文件，需要调用一个不同的方法。pygal 提供了几个选项，对应不同的文件类型：

```
worldmap_chart.render_to_file('world_map.svg')

worldmap_chart.render_to_png('world_map.png')
```

现在打开 .svg 或 .png，会看到图 10-5。

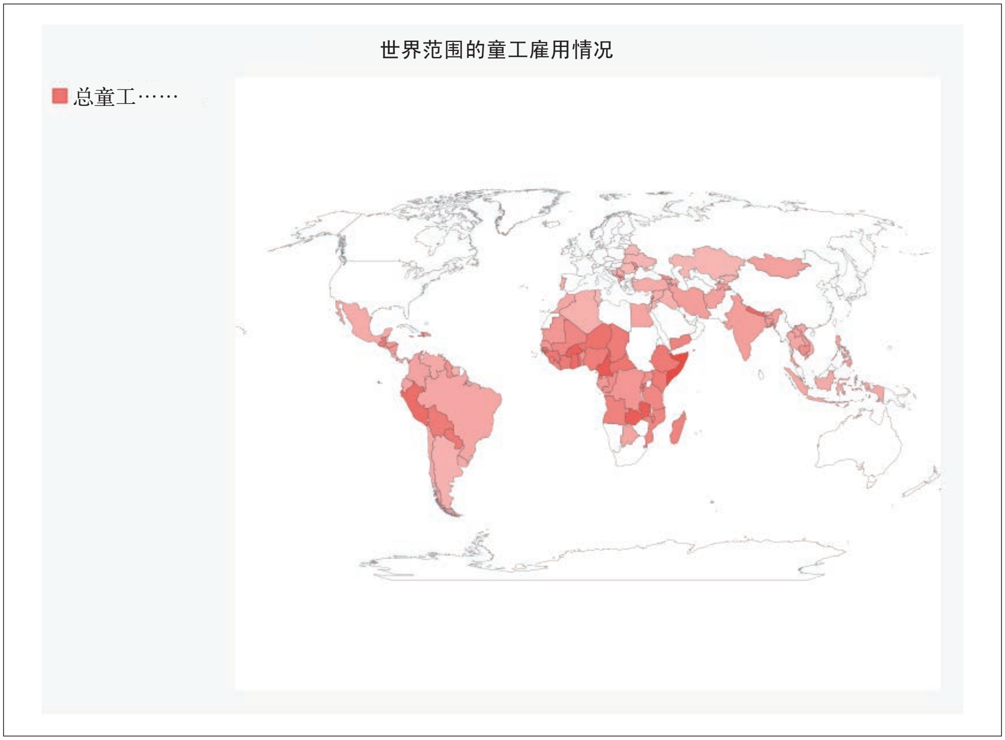


图 10-5：世界地图

> 如果在渲染地图时有任何的问题，确保所有的依赖库已经安装。如果电脑上没有 .svg 文件查看器，你可以在浏览器中打开 .svg 文件，像图 10-5 一样。

强烈建议你查看除 .svg 之外 `pygal` 提供的其他选项。文档中有很多实例，高级的和简单的都有，对初学者来说，它也是一个很容易使用的 .svg 库。

10.2.4 交互式元素

交互式元素通过网站交互或模拟讲故事。因为用户可以通过浏览器四处点击和探索，所以他们可以以自己的节奏了解这个主题，从数据中得出自己的结论。这对于需要更加深入研究才能充分理解的主题特别有用。

作为对最近在美国麻疹疫情爆发的回应，英国《卫报》制作了一个疫情爆发交互式网页（https://www.theguardian.com/society/ng-interactive/2015/feb/05/-sp-watch-how-measles-

outbreak-spreads-when-kids-get-vaccinated），允许用户查看和回放拥有不同疫苗接种率的各地区爆发潜在麻疹疫情的影响。这个交互式网页展示了《卫报》工作人员研究和编码的不同场景。不是每一种模拟都得到相同的输出，用户可以看到有一个展示接种率的控件，并且同时展示了患病概率（即，更高的疫苗接种率会有更小的感染可能）。这个网页使用了一个高度政治化的主题，并且使用疫情爆发的统计学模型呈现出了真实世界的场景。

尽管交互式元素需要更多的经验来创建，并且经常需要更高的编码技能，但它们是非常棒的工具，特别是当你有前端编码经验的时候。

举个例子，对于童工数据，可以创建一个交互式页面，用于展示在乍得本地高中，有多少人由于童工雇用率而可能永远不会毕业。另一个交互式页面可以展示当地商场中利用童工生产或提供的商品和服务。交互式元素将难以可视化的信息展示给受众，这样他们可以理解数据，与你的故事建立联系。

10.2.5 文字

通过文字讲故事，对作者和记者来说是非常自然的。无论你使用哪一种视觉方式，使用的所有文字对目标听众应该有用且合适。你可能想要采访该领域的专家，或同他们交谈。引用他们关于该发现的语言、想法和结论，会帮助听众综合这些信息。

如果正在研究一个本地的学校董事会是如何确定下一个学年的预算的，你可以同董事会成员谈话，或许还能得到有关提议修改的内部信息。如果正在研究公司准备发布的新产品，你可能想要同一些关键决策者谈话，确定可能会有什么新产品。

如果想获得更多关于访谈以及如何使用引言来完善故事的内容，Poynter 针对如何成为一名更好的访谈者提出了一些很棒的建议（http://www.poynter.org/2013/how-journalists-can-become-better-interviewers/205518/），哥伦比亚大学的访谈准则（http://www.columbia.edu/itc/journalism/isaacs/edit/MencherIntv1.html）分享了一些关于如何准备访谈以及如何根据项目需要准备不同的访谈的建议。

> 如果你是一个领域的专家，使用了一些技术或不熟悉的术语，依照听众的情况，你可能需要将这些主题分解为一个个小块。一个简单的术语表会很有用。对于科学、技术和医疗著作，当它们面向的是更广泛的读者时，使用术语表是非常普遍的做法。

10.2.6 图片、视频和插画

如果你的故事有一个很强的视觉元素，图片和视频可以增强这个故事。例如，与主题相关人物的视频访谈可以展示数据的个人立场，并且可能揭开了研究中其他的视角或未来的方向。

和视频一样，图像可以为观众描绘画面。根据我们的经验，使用图形图像来描绘战争或其他可怕的事情，总会影响我们对故事的解释。但是，使用图片来简单地震撼观众会导致他们忽略你为工作所做的细致研究。思考一下这一点，为你的故事找到一个平衡点。

如果你无法获得与主题相关的照片或者没有能力自己收集，可以用插画讲故事。《华盛顿邮报》一篇关于健康与不健康办公作空间的报道（http://www.washingtonpost.com/wp-srv/special/health/unhealthy-vs-healthy-office/），使用了一张插画来展示故事的概念。

对于童工数据来说，我们不大可能有机会自己收集在数据分析中发现的罪行的视频和照片。然而，我们可以使用过去关于童工报道的图片（在经过允许并说明来源的情况下），来揭示儿童仍然遭受着这个世界性问题的影响。

10.3　展示工具

如果你不想发布数据，但是想将其展示给一小群人（或内部人员），创建一个幻灯片相对更简单一些。在展示数据有很多方式可选时，你可以创建一个灵活的幻灯片，而不需要做太多额外的工作。

一个拥有很高评价的幻灯片制作工具是 Prezi（https://prezi.com/），它可以创建看起来很专业的幻灯片。Prezi 让你能够创建公共可用的幻灯片，而且它有多种不同的桌面客户端（如果你想要创建私密幻灯片，需要注册一个付费账户）。Haiku Deck（https://www.haikudeck.com/）是另一个只可在线使用的网站工具，可以免费制作公开的幻灯片，付费制作私密的幻灯片。你还可以将 Google Slides 作为一个免费且简单的选择，特别是当你准备给内部人员演示并且你的公司也使用 Google Apps 的时候。

10.4　发布数据

你已经花费了时间研究、探索和展示数据，现在想要通过网络分享报告给全世界。当你准备在网络上发布数据时，首先应该确定数据是否能够被公开访问。



如果你的展示包含私密的数据或者数据只与你的公司相关（专利数据），你应该在一个受密码保护的网站上发布它，或者在内网发布。

如果你想要和世界分享数据，通过诸多网络平台之一来发布不会有问题。在这一节里，我们会介绍如何在易于使用的免费博客平台或者你自己的网站上发布数据。

10.4.1　使用可用站点

许多网站被设计用来发布数据，以迎合类似你这样的想要分享报告或想法并将它们简单地发布到网络上的作者和研究者。下面是一些较好的选择。

1. Medium

在 Medium（https://medium.com/）上，你可以创建一个账户，开始写自己的文章，轻松地嵌入评论、引用、图片和图表。因为这是一个社交媒体平台，所以其他的 Medium 用户可以推荐你的文章，分享它，标记它，同时关注你之后的文章。

使用一个类似 Medium 的托管站点让你能够专注于写作和报告，不用花时间去琢磨如何搭建和维护你自己的网站。

Medium 团队维护了一些很好的图表工具，包括 Charted.co（https://github.com/mikesall/charted），它使用简单的 CSV 或 TSV 文件来渲染一个交互图表。在写作本书时，它们还没有实现直接嵌入这些图表到文章中的能力，但是很有可能会添加这个功能。

Medium 使得将不同种类的社交媒体、视频、照片和其他媒介嵌入文章很容易（https://medium.com/@Medium/embed-videos-tweets-music-and-more-into-your-medium-stories-3b5c09c116e8#.w5e3j4n9v）。你可以通过阅读 Medium 每月最受欢迎的文章（https://medium.com/top-100/）来获得很多很棒的关于讲故事的想法。

建议阅读和搜索你主题领域的 Medium 博文，同这一主题的其他作者联系，来了解他们是如何讲故事的。

Medium 是在社交网络上写作并将你的想法分享给世界的很好方式。但是如果你想要运行自己的博客呢？继续阅读下面关于搭建和运行网站的选择。

2. 快速上手的网站：WordPress、Squarespace

如果想要对布局和内容有更多的控制，你可以在 Squarespace（http://www.squarespace.com/）或 WordPress（https://wordpress.com/）上搭建自己的博客。这两个平台给了你一个免费（WordPress）或者廉价（Squarespace）的被维护的站点，让你可以自定义站点的外观和风格。你可以设置一个域名，这样你的文章会挂载在自己的 URL 下。

大多数虚拟主机提供商为 WordPress 开发了一键安装版本供你使用。你需要选择一个用户名和一些站点标题，并且确保有一个足够强大和安全的密码。在 WordPress 里，你有很多主题（https://wordpress.org/themes/browse/popular/）和插件（https://wordpress.org/plugins/browse/popular/）可以选择，来自定义网站的样式、风格和功能。为了保护站点，建议你安装一个流行的安全插件，并阅读 WordPress 关于安全的建议（http://codex.wordpress.org/Hardening_WordPress）。

使用 Squarespace，只需要注册一个账户，选择一个布局。你可以自定义相关联的社交媒体、域名，以及你是否想要有一个电商店铺。

一旦站点就绪，并且运行起来，添加内容是非常简单的。你会想要发布新的页面或文章，使用内置的编辑器添加文字和图片（或者，如果你正在使用 WordPress，可以安装额外的编辑器插件，支持更多的特性），之后发布内容。

你可以让文章更容易被找到，方法是通过 SEO 花费一些时间来填充描述和关键词，来提高文章的可见度。WordPress 插件和 Squarespace 特性可以为每篇文章做到这些。

3. 自有博客

如果你运行着自己的网站或博客，你已经有了一个很棒的分享报告的平台。你需要确保可以适当地将视觉故事嵌入到网站中。生成的大多数图表可以很容易地嵌入到网站的 HTML 代码中。

如果你在使用除了 WordPress 或 Squarespace 以外的其他平台，可能需要研究在站点上如何分享图表、视频和照片。建议你联系平台的社区或创建者，或者阅读站点的指引和文档，来确定如何最好地嵌入图片、图表和交互式元素。

10.4.2 开源平台：创建一个新网站

我们已经提到了几个使用 Squarespace 和 WordPress 等免费或廉价平台创建和运行新站点的选项；但是如果你想要启动、运行和维护自己的站点，可以从众多伟大的开源平台中选择一个。

1. Ghost

Ghost 是一个很容易运行的平台（https://github.com/tryghost/Ghost）。Ghost 使用 Node.js（https://nodejs.org/），一个开源的 JavaScript 异步服务器；如果你对 JavaScript 很感兴趣，使用和学习它都很有趣。因为它是异步的，所以拥有很好的性能，能够处理大量的请求。Ghost 还提供了搭建托管站点的能力（https://ghost.org/），与 WordPress 或 Squarespace 类似，收取一定的费用。

如果你想要挂载自己的 Ghost 博客，DigitalOcean 和 Ghost 合作提供了一个容易使用和安装的服务器镜像（https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-ghost-application），不用一个小时就可在你的服务器上创建和运行 Ghost 站点。如果这是你第一次搭建服务器，强烈建议使用这个方式，因为一些初始的工作已经替你完成。

如果你有自己的服务器，并且想要在其他的平台上从头开始安装 Ghost，Ghost 提供了一些教程（http://support.ghost.org/deploying-ghost/）。你需要执行以下主要步骤。

(1) 下载并安装最新的源代码。
(2) 运行 `node`［建议你使用 `nvm`（https://github.com/creationix/nvm）］。
(3) 使用 `npm`（node 版本的 pip）安装 `node` 依赖。
(4) 运行 `pm2`（https://github.com/Unitech/pm2）来管理 Ghost 进程。
(5) 启动 nginx，使用网关与运行的 Ghost 进程通信。
(6) 开始写博客！

如果你遇到了问题，可以登入 Ghost 的 slack 频道（https://ghost.org/slack/），看看是否有人能够帮助你，或者在 Stack Overflow（http://stackoverflow.com/）上搜索更多相关的信息。

2. GitHub Pages和Jekyll

如果你使用 GitHub 托管代码，你也可以用它来托管自己的网站。GitHub Pages（https://pages.github.com/）是一个依靠 GitHub 运行的网站托管工具，让你可以灵活地部署，并能轻松创建内容。使用 GitHub Pages，你可以通过将静态内容提交到你的仓库，直接部署你

的 GitHub 页面。如果你喜欢使用框架，可以使用 Jekyll（http://jekyllrb.com/），一个基于 Ruby 的静态页面生成器，它集成了 GitHub Pages。

Jekyll 的文档（http://jekyllrb.com/docs/home/）有一个说明性的概述，涵盖了如何在本地安装和运行 Jekyll，但是建议你阅读 Barry Clark 为 *Smashing* 杂志编写的文章（https://www.smashingmagazine.com/2014/08/build-blog-jekyll-github-pages/）。在这篇文章中，他阐述了如何复制（fork）一个已有仓库，运行自己的站点，以及修改 Jekyll 的设置和特性。如果你不想使用 Jekyll，但是仍然想使用 GitHub Pages，你可以使用库或者手动地生成静态的 HTML 文件，再将这些文件提交到 GitHub Pages 仓库。

> 一个很容易使用的 Python HTML 生成器是 Pelican（https://github.com/getpelican/pelican），它可以接受 AsciiDoc、Markdown 或者 reStructuredText 格式的文件，并将它们转化为静态内容。它提供了简单的步骤来启动评论和访问分析，以及使用 GitHub Pages 的相当全面的介绍（http://docs.getpelican.com/en/latest/tips.html#publishing-to-github）。

还有很多其他的静态站点生成器，也还有很多关于如何将它们与 GitHub Pages 集成的文章。一个搭建 GitHub Pages 博客的选择是 Hexo（http://jdpaton.github.io/2012/11/05/setup-hexo/），它是一个基于 Node.js 的框架。Octopress（https://github.com/octopress/octopress）是另外一个很棒的选项，它基于 Jekyll 构建，所以你可以轻松地使用 GitHub Pages 和 Ruby 来发布和部署站点。

3. 一键部署

如果你坚持使用大型的博客工具或网站框架，比如 WordPress，DigitalOcean 有很多一键安装的包（https://www.digitalocean.com/features/one-click-apps/），让你能够在短时间内搭建自己的服务器，并安装所有必需的库和数据库。它同样提供了便捷的入门指引，描述了如何在 droplet[2]）上安装 WordPress（https://www.digitalocean.com/community/tutorials/how-to-use-the-wordpress-one-click-install-on-digitalocean）。

除了大型的虚拟主机提供者外，你同样可以在 Heroku（一个基于云的应用主机服务商，https://devcenter.heroku.com/start）上使用 Python、Ruby 和其他开源的平台。如果你正在使用或学习一个开源框架，可以使用 Heroku 来部署自己的网站；它提供了很棒的文档和技术支持。

无论你使用哪一个框架或解决方案，专注于用简单的方式在网络上发布内容或代码，是很重要的。选择一些简单直接的方案，并专注于恰当地向全世界展示、发布和分享内容。

10.4.3　Jupyter（曾名IPython notebook）

我们已经介绍了怎样分享你的发现，但是如果你还想分享代码、数据或者研究过程呢？根据听众的不同，分享代码并允许人们直接与其交互可能是很合适的。如果你准备分享给同事和同行，这是一个很好的展示你如何进行研究的方式。

注 2：DigitalOcean 虚拟主机的别名。——译者注

Jupyter notebook（https://jupyter.org/，曾名 IPython notebook，http://ipython.org/notebook.html）是一个很好的分享 Python 代码和代码生成的图表的方式。这些 notebook 组合了易于使用的浏览器和 IPython 的交互特性。notebook 在迭代代码设计和数据探索中也非常有用。

> 正在学习新的库或者使用新数据？在 Jupyter notebook 中保存你的工作。一旦完成了迭代并优化了代码，你可以将代码中重要的部分移动到仓库中，恰当地结构化、文档化，将这些东西综合在一起。

使 Jupyter 就绪并在本地运行它非常简单，只需运行这个命令：

```
pip install "ipython[notebook]"
```

要启动 notebook 服务器，运行：

```
ipython notebook
```

你看到的终端输出应该类似于：

```
[NotebookApp] Using MathJax from CDN: https://cdn.mathjax.org/mathjax/latest/
    MathJax.js
[NotebookApp] Terminals not available (error was No module named terminado)
[NotebookApp] Serving notebooks from local directory: /home/foo/my-python
[NotebookApp] 0 active kernels
[NotebookApp] The IPython Notebook is running at: http://localhost:8888/
[NotebookApp] Use Control-C to stop this server and shut down all kernels.
Created new window in existing browser session.
```

这是 notebook 服务器启动的过程。你会看到一个新的浏览器窗口（或 tab）打开一个空的 notebook。

根据运行 notebook 文件夹的不同，你可能会在浏览器中看到一些文件。notebook 服务器直接在当前文件夹中运行，并展示这个文件夹的内容。建议为 notebook 创建一个新的文件夹。为了创建新的文件夹而停止服务器，需在运行的终端中输入 Ctrl-C（Windows 和 Linux 上）或者 Cmd-C（Mac 上）。创建一个新的目录，切换目录到这个文件夹下，重新启动服务器，类似下面这样：

```
mkdir notebooks
cd notebooks/
ipython notebook
```

让我们通过创建一个新的 notebook 来使用 Jupyter。为了达到这个目的，点击 New 下拉菜单，选择 Notebooks 头部下的 Python 2。创建好新的 notebook 后，给它一个有用的名称。为此，点击 title 区域（这里当前应该为“Untitled”），输入一个新名称。为 notebook 命名会在将来节省你大量的搜索时间。

在 Jupyter 中，每一个文本区域被叫作单元。notebook 支持多种不同的单元类型。在顶部和代码间使用一些 Markdown（https://daringfireball.net/projects/markdown/syntax）单元来解释并给代码添加文档是一个很好的想法。图 10-6 展示了一个添加头部（header）的示例。

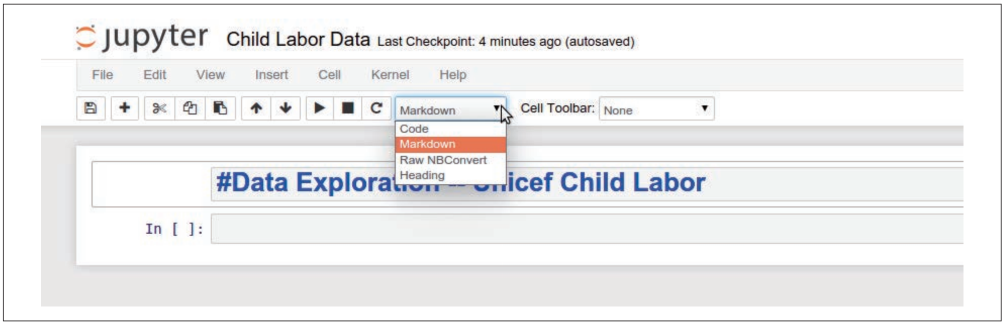


图 10-6：添加 Markdown 标题

要开始编写 Python，只需点击下一个可用的单元，然后输入即可。当你完成了编写的语句或函数后，敲击 Shift+Enter。代码会执行并出现一个新的单元，在这里你可以编写下一个 Python 代码。正如在图 10-7 和你自己的 notebook 中看到的那样，你可以看到在一个普通的 Python 解释器中会看到的所有输出。

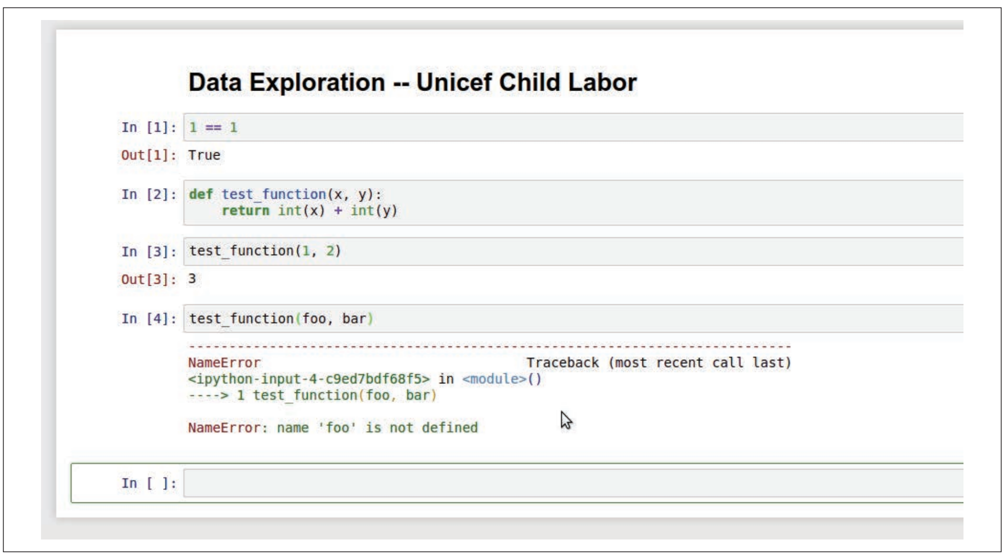


图 10-7：在 Jupyter 中工作

有很多非常棒的 Jupyter（和 IPython）notebook 指南，但是一个很好的入手点可能是重新尝试一些本书中使用过的代码。

建议组织你的 notebook，使其类似于你的仓库。你可能希望根目录下有一个包含数据的数据（data）文件夹，以及一个包含可导入 notebook 的脚本的工具（utils）文件夹。你的 notebook 就像另一个脚本，只是它是交互式的，并且在浏览器中。

用完 notebook，点击保存按钮（确保它创建一个新的检查点，这样可以更新你的文件）。

如果你在一个特定的 notebook 中完成了工作，但是仍在使用其他的 notebook，停止老的 notebook 进程是明智的选择。为此，选择服务器上的 Running 标签，并点击 Shutdown 按钮。当你对所有的 notebook 完成了编辑，保存它们并使用 Ctrl-C 或 Cmd-C 在运行 notebook 的终端停止服务器。

共享的Jupyter notebook

现在你已经熟悉 Jupyter notebook 的使用了，可以通过共享服务器上传并分享代码。这使得他人可通过普通网络（不仅是本地主机，例如前文在你的终端上运行的 notebook）访问你的 notebook。

有一些很棒的入门教程说明了如何使用 DigitalOcean（http://calebmadrigal.com/ipython-notebook-vps/）、Heroku（https://github.com/mietek/instant-ipython）、Amazon 网络服务（https://gist.github.com/iamatypeofwalrus/5183133）、Google DataLab（https://cloud.google.com/datalab/），或者你喜欢的任意服务器（http://ipython.org/ipython-doc/1/interactive/public_server.html#notebook-public-server），来搭建一个 notebook 服务器。

> 记得在 notebook 服务器上使用安全密码，保证 notebook 只被有这个密码的人使用。这会确保服务器和数据安全。

建议你也为 Jupyter notebook 建立一个类似 Git（第 14 章再深入探索）这样的版本控制系统，这样你就有了 notebook 每天或每周的历史记录。通过这种方式，你可以恢复删除的东西，同时帮你保存和组织代码。

> 如果你正在使用一个共享的 notebook 服务器，确保人们知道内核被中断（这在服务器重启或者有人终止或重启 notebook 内核时都会发生）时如何运行所有的代码。为了运行所有 notebook 代码，选择 notebook 工具栏中的 Cell 下拉菜单，然后点击 Run All 按钮。你也应该建议用户在完成工作后使用 Shutdown 终止 notebook，这样服务器上就没有无用的运行进程。

无论本地还是共享的 Jupyter notebook 都是展示数据和工作流的很好的工具。当你回顾数据探索和分析时，在本地运行它们将会非常有用。随着 Python 知识的增长，你可以将脚本迁移到 Python 3，同时运行 JupyterHub（https://github.com/jupyter/jupyterhub）。JupyterHub 就一个多用户的 notebook 服务器，运行着多种语言（包括 Python），当前正处在积极地开发中。

无论选择在 notebook 服务器还是开源平台上发布，你现在已经掌握了技能，能够分析如何以最佳方式展现和发布你的发现、数据和代码。

10.5　小结

你已经学习了如何将数据转化为可演示的形式，并且通过互联网传播它。你有很多发布选择，它们有不同的隐私等级和维护需求。你可以为报告搭建一个网站，并创建美丽的图片

和图表来讲述故事。有了 Jupyter 的帮助，你可以很容易地分享和演示你写的代码，同时交他们一点 Python 知识。

你同样学习了表 10-1 中列出的库和概念。

表10-1：新的Python和编程概念和库

概念/库	功能
用于绘图的 matplotlib 库	可以使用两个图表库生成简单的图表。你可以为图表使用标签和标题，用更清楚的方式展示数据
用于更复杂图表的 Bokeh 库	允许你轻松地生成更复杂的图表，可以在图表中添加交互式元素
用于 SVG 图形和地图的 pygal 库	pygal 让你能够使用简单的函数传递数据生成 SVG 图片
Ghost 博客平台	基于 Node.js 的博客平台，让你能够在自己的服务器（或挂载在 Ghost 上的平台）上快速地构建一个博客，在自己的网站上分享故事
GitHub Pages 和 Jekyll	一个集成在 GitHub 上的简单发布平台，你可以通过简单地提交代码到仓库中来分享文章和演示文档
Jupyter notebook	一个同其他开发者或同事分享代码的简单方式，同样也是一种使用敏捷开发方法（反复试错）开始开发自己的代码的很好的方式

接下来，我们会学习如何通过网络爬虫和 API 收集更多的数据。你在本章学到的知识会用在今后收集的数据上，所以请继续阅读，学习新的演示技巧。在后面的章节里，你会学得更高级的 Python 数据技术，让你更好地使用 Python 收集、评估、保存和分析数据。你在本章学到的讲故事工具会帮助你更好地进行数据处理，将研究所得分享给听众和全世界。

第 11 章

网页抓取：获取并存储网络数据

网页抓取是当今世界数据挖掘中必不可少的一部分，因为你几乎可以在网络上找到任何事物。有了网页抓取，你可以使用 Python 库来探索 Web 页面、搜索信息并收集它们以撰写报告。网页抓取让你爬取站点，发现在没有机器人协助的情况下不容易获取的信息。

这项技术使你能够获取 API 或文档之外的数据。想象一个脚本登录你的 E-mail 账户，下载文件，运行分析，并且发送一个整合的报告。想象一下不用使用浏览器就可以测试站点，以确定它具备完整的功能。想象一下从一个定期更新的网站的一系列表格中抓取数据。这些示例展示了网页抓取如何能帮助你完成数据处理的需求。

根据爬取内容的不同——本地或公开站点，XML 文档——你可以使用很多相同的工具完成这些任务。大多数网站在 HTML 代码中包含数据。HTML 是一种标记语言，使用括号（类似于第 3 章中的 XML 示例）来包含数据。在这一章，我们会使用一些能够解析和读取 HTML 和 XML 等标记语言的库。

很多站点使用内部的 API 和嵌入的 JavaScript 脚本来控制页面上的内容。由于这些构建站点的新方式，并不是所有的信息都能够使用读页面的抓取器找到。我们还会学习如何使用一些读屏幕的 Web 抓取器，应对拥有多个数据源的站点。根据站点的组成，你可能同样可以连接 API；在第 13 章你会了解更多有关 API 的信息。

11.1　抓取什么和如何抓取

网页抓取为数据收集带来了无限可能。在互联网上有成千上万的站点，拥有可能会在项目中使用的各种各样的内容和数据。为了构建一个认真负责的网页抓取器，要熟悉每一个站点，以及可抓取的内容。

版权、商标和抓取

当在网络上抓取时，对于你找到的所有媒体（来自报纸、杂志、书籍或博客），你应该考虑收集的数据和它们的使用方式。你是否会下载其他人的照片并且将它当作自己的照片发布？不，这是不道德的，而且在一些情况下是非法的。

学习像版权（http://www.dmlp.org/legal-guide/copyright）和商标（http://www.dmlp.org/legal-guide/trademark）这样的媒体法会影响你的决定，尤其是要抓取的数据属于某人的知识产权（http://www.dmlp.org/legal-guide/intellectual-property）时。

研究域名并查阅法律允许内容和禁止内容的有关提示，还要熟读 robots 文件（http://www.robotstxt.org/robotstxt.html）来更好地理解网站所有者的意愿。如果你不确定数据能否被抓取，可联系律师或网站本身。取决于你的住址和使用数据的目的，如果你对本国法律和判例存有疑问，可能需要联系一家数字媒体法定组织。

对于大多数的网络抓取，抓取文本会比抓取链接、图片或图表更合理。如果你还需要保存链接、图片或文件，这其中的大多数都可以使用简单的 bash 命令（例如 `wget` 或 `curl`，http://www.thegeekstuff.com/2012/07/wget-curl/）下载，而这不需要 Python。你可以直接保存一个 URL 列表到文件中并且写一个脚本来下载文件。

我们从简单的文本抓取开始。大多数网页的构建都基于适当的 HTML 标准，结构相似。大多数的网站有一个头部，大多数的 JavaScript 和页面样式文件在这里定义，同时还有其他额外信息，比如类似 Facebook、Pinterest 这样的服务的元标签，以及搜索引擎用法的描述信息。

头部之后是主体。主体是站点的主要部分。大多数的站点使用容器（类似 XML 节点的标记节点）来组织站点，并且允许站点内容管理系统加载内容到页面中。图 11-1 展示了一个典型的网页是如何组织的。

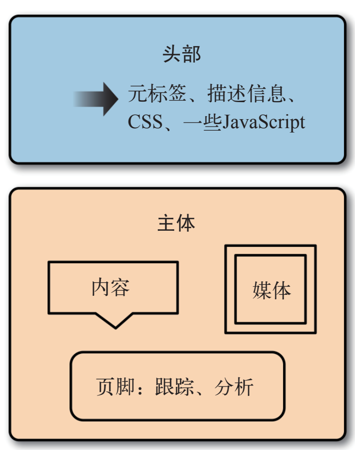


图 11-1：网页解剖

对于很多站点来说，页面的顶部部分包含到站点主要部分或者相关主题的导航和链接。链接或者广告通常出现在页面两边向下延展的位置。页面的中间部分通常包含你想要抓取的内容。

> 熟悉大多数网页的结构（元素的视觉位置和它们在标记语言中的位置）会帮助你从互联网上抓取数据。如果可以聚焦到数据源，你就可以快速地构建抓取器。

一旦知道了在页面上寻找什么，并且通过学习页面源代码的结构分析了页面的组成，你就可以确定如何收集页面中的重要的部分。许多网页在第一次页面加载的时候提供内容，或者提供一个已加载好内容的缓存页面。对于这些页面，可以使用简单的 XML 或 HTML 解析器（我们会在本章学习它们），并且从第一个 HTTP 响应（在你请求一个 URL 时浏览器加载的内容）中直接读取内容。这与读取文档类似，只是需要一个初始的页面请求。

如果你需要首先同页面交互来获取数据（也就是输入数据和点击按钮），并且它不仅仅是一个简单的 URL 的改变，你需要使用一个基于浏览器的抓取器，在浏览器中打开页面同它交互。

如果需要遍历整个网站来收集数据，你会想要一个*爬虫*：一个机器人，它爬取网页，并且根据规则识别好的内容或跟踪更多页面。我们在爬取中使用的库非常地快速、灵活，让编写这些类型的脚本变得十分简单。

在开始编写抓取器代码之前，我们会查看一些网站，习惯于分析要使用那个类型的抓取器（页面读取器、浏览器读取器或爬虫），以及抓取数据会多难或多简单。有时，确定数据值得付出多少努力是很重要的。我们会介绍一些工具来确定为抓取数据需要付出多少努力，以及值得为这项工作投入多少时间。

11.2　分析网页

网络抓取的大多数时间会花费在观察浏览器标记语言和搞清楚如何同它交互上。了解你最爱的浏览器的调试或开发工具是成为一名高级网页抓取者的必要环节。

根据浏览器的不同，工具可能有不同的名称和功能，但是概念是相同的。你需要自学最喜欢的浏览器工具，不管是 IE（https://msdn.microsoft.com/library/bg182326(v=vs.85)）、Safari（https://developer.apple.com/safari/tools/）、Chrome（https://developer.chrome.com/devtools）或 Firefox（https://developer.mozilla.org/en-US/docs/Tools/GCLI）。

每个浏览器的调试器都是类似的。你会在一个区域看到请求和页面加载数据（通常叫网络或者其他类似的东西）；在另外一个区域分析页面的标记信息，看到每个标签中的内容和样式（通常叫作检视、元素或 DOM）。在第三个区域，你可以看到 JavaScript 错误，并同页面中的 JavaScript 交互；这个区域通常叫作控制台。

你的浏览器开发者工具可能还有其他的标签，但是我们真的只需要这 3 个标签来理解页面是如何构建的，以及如何简单地抓取内容。

11.2.1 检视：标记结构

当你想要抓取一个站点的时候，首先分析站点结构和标记语言。像在第 3 章学到的那样，XML 的结构由节点和内容以及键和值组成。HTML 非常相似。如果打开浏览器的开发者工具，浏览检视（Inspection）、元素（Elements）或 DOM 标签，你会看到一系列的节点和它们的值。节点和其包含的数据同我们在 XML 示例中看到的有一些不同——它们是 HTML 标签（表 11-1 列出了一些基本的标签）。HTML 标签用来告诉你内容信息。如果你想找到页面上的所有图片，查找 `img` 标签。

表11-1：基本的HTML标签

标签	描述	示例
`head`	用来存储元数据和文档的其他必需信息	`<head> <title>Best Title Ever</title> </head>`
`body`	用来存储页面大部分内容	`<body> <p>super short page</p> </body>`
`meta`	用来存储元数据，例如站点简短的描述或关键词	`<meta name="keywords"content="tags, html">`
`h1`，`h2`，`h3` …	用来存储头部信息；数字越小，头部越大	`<h1>Really big one!</h1>`
`p`	用来存储文本段落	`<p>Here's my first paragraph.</p>`
`ul`，`ol`	用来存储无序表（`ul`：圆点）和有序表（`ol`：数字）	`<ul><li>first bullet</li></ul>`
`li`	用来存储列表对象；应该始终位于一个列表（`ul` 或 `ol`）中	`<ul><li>first</li> <li>second</li></ul>`
`div`	用于分节或划分内容	`<div id="about"><p>This div is about things.</p></div>`
`a`	用于链接内容，被称作“锚标签”	`<a href="http://oreilly.com">Best Ever</a>`
`img`	用于插入一张图片	`<img src="/flying_cows.png" alt="flying cows!" />`

关于 HTML 标签和其使用方式的完整介绍，请查看 Mozilla 开发者网络的 HTML 参考、指南和介绍（https://developer.mozilla.org/en-US/docs/Web/HTML）。

除了使用的标签和内容结构，每个标签之间放置的位置很重要。类似于 XML，HTML 也有*父元素*和*子元素*。在结构中存在层次关系。父节点拥有子节点，而学习如何遍历家族树结构会帮助你得到想要的内容。了解元素之间的关系，无论它们是双亲节点、子节点还是同级节点，会帮助你编写更高效、快速和易于更新的抓取器。

让我们仔细地查看在 HTML 页面中这些关系意味着什么。下面是一个基本的 HTML 站点的结构。

```
<!DOCTYPE html>
<html>
<head>

  <title>My Awesome Site</title>
  <link rel="stylesheet" href="css/main.css" />
```

```
</head>
<body>
   <header>
        <div id="header">I'm ahead!</div>
   </header>
   <section class="main">
    <div id="main_content">
        <p>This site is super awesome! Here are some reasons it's so awesome:</p>
                <h3>List of Awesome:</h3>
                <ul>
                    <li>Reason one: see title</li>
                    <li>Reason two: see reason one</li>
                </ul>
        </div>
   </section>
   <footer>
        <div id="bottom_nav">
           <ul>
                <li><a href="/about">About</a></li>
                <li><a href="/blog">Blog</a></li>
                <li><a href="/careers">Careers</a></li>
           </ul>
        </div>
   <script src="js/myjs.js"></script>
   </footer>
</body>
</html>
```

如果从这个页面的第一个标签开始（文档类型声明下），可以看到整个页面的所有内容都在 html 标签下。html 标签是整个页面的根标签。

在 html 标签内，有标签 head 和 body。页面的大部分内容在标签 body 内，但是 head 也有一些内容。标签 head 和 body 是 html 元素的子标签。反过来，这些标签有着他们自己的子标签和后继标签。head 和 body 标签是同级关系。

查看主 body 标签的内部，可以看到其他一些家族关系。所有这些列表对象（li 标签）是无序列表（ul 标签）的子标签。header、section 和 footer 标签是同级关系。script 是 footer 的子标签，是 footer 中 div 标签的邻居，用来存储链接。还有很多复杂的关系，这只是一个简单的页面！

为了更深入地研究，下面的代码展示了一个有着更复杂关系的页面（处理网页抓取时，几乎很难有一个所有元素组织合理且关系完整的完美页面）：

```
<!DOCTYPE html>
<html>
    <head>
        <title>test</title>
        <link ref="stylesheet" href="/style.css"/>
    </head>
    <body>
        <div id="container">
            <div id="content" class="clearfix">
                <div id="header">
```

```
                <h1>Header</h1> ❶
            </div>
            <div id="nav"> ❷
                <div class="navblock"> ❸
                    <h2>Our Philosophy</h2>
                    <ul>
                        <li>foo</li>
                        <li>bar</li>
                    </ul>
                </div>
                <div class="navblock"> ❹
                     <h2>About Us</h2> ❺
                     <ul>
                         <li>more foo</li> ❻
                         <li>more bar</li>
                     </ul>
                </div>
                </div>
                <div id="maincontent"> ❼
                    <div class="contentblock">
                        <p>Lorem ipsum dolor sit amet...</p>
                    </div>
                    <div class="contentblock">
                        <p>Nunc porttitor ut ipsum quis facilisis.</p>
                    </div>
                </div>
            </div>
        </div>
        <style>...</style>
    </body>
</html>
```

❶ 当前元素父元素的前面的邻居的第一个子元素。
❷ 当前元素的父级 / 祖先。
❸ 当前元素的邻居。
❹ 当前元素。
❺ 当前元素的第一个子元素 / 后代。
❻ 当前元素的子元素 / 后代。
❼ 当前元素父元素的下一个邻居。

为便于讨论，“当前元素”是第二个为 navblock 类的 div。可以看到它有两个子元素，一个是标题（h2），另一个是无序列表（ul），同时还有列表对象（li）位于列表中。它们是后代（取决于你想使用的库，它可能被包含在“all children”中）。当前元素有一个邻居，即第一个为 navblock 类的 div。

ID 为 nav 的 div 是当前元素的父元素，但是我们的元素有其他的祖先。如何从当前元素移动到 ID 为 header 的 div？我们的父元素是 header 元素的邻居。为了得到 header 元素的内容，可以找到父元素的前面的邻居。父元素同样有另外一个邻居，即 ID 为 maincontent 的 div。

所有这些关系被描述为文档对象模型（DOM）结构。HTML 用规则和标准来组织页面上的内容（也被称作文档）。HTML 元素节点是“对象”，并且为了正确地展示，它们必需遵循一个模型 / 标准。

花费在理解节点之间的关系上的时间越多，使用代码快速有效地遍历 DOM 树就越容易。在本章之后的部分，我们会介绍 XPath，它使用家族关系选择内容。现在，进一步理解了 HTML 结构和 DOM 元素之间的关系之后，我们可以更仔细地研究定位和分析在选择的站点上想要抓取的内容。

你或许可以使用开发者工具搜索标记代码，这取决于浏览器。这是一种非常棒的查看元素结构的方式。举个例子，如果正在寻找内容中一个特别的小节，可以搜索这些词语，找到它们的位置。许多浏览器还允许你右击页面上的元素，选择“检视”。这通常会打开你的浏览器工具来选择元素。

我们会在示例中使用 Chrome，但是你可以使用自己最喜欢的浏览器。当研究非洲的童工时，我们发现了将童工与冲突相联系的数据。这促使我们找到致力于阻止非洲冲突地带和冲突矿产的组织。打开其中一个组织的页面：Enough 项目的 Take Action 页面（http://www.enoughproject.org/take_action）

当第一次打开开发者工具时——在 Chrome 中选择工具→开发者工具，在 IE 中敲击 F12，在 Firefox 中选择工具→网页开发者→检视器，或者在 Safari 高级设置中启用开发者菜单——我们会在一个面板中看到标记信息，在另外一个小的面板中看到 CSS 规则和样式，在工具上方的一个面板中看到真正的页面。对于不同的浏览器，布局可能会不同，但是用这些工具查看这些特性时应该会很相似（见图 11-2）。

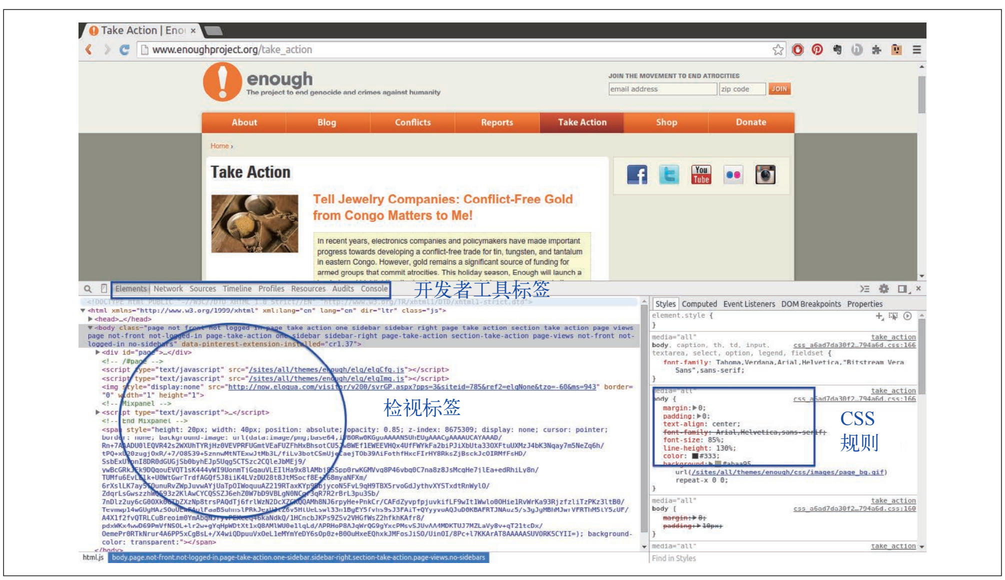


图 11-2：Enough 项目的 Take Action 页面

如果移动光标到开发者工具的标记部分（检视标签），你可能会看到页面上不同的部分出现高亮。这是一个非常棒的特性，可以帮助你看到标记和页面结构中不同的元素。

如果点击与 `div` 和页面主元素相邻的箭头，你可以看到位于其中的元素（子元素）。举个例子，在 Enough 项目主页上，可以通过点击右边栏（在图 11-3 中圈出），打开 `main-inner-tse div` 和其他内部的 `div`。



图 11-3：探索侧边栏

可以看到侧边栏图片在链接之内，链接位于一个段落之中，而段落在 `div` 中——这个列表很长。理解什么时候图片在链接中（或不在），确定哪个内容位于段落标签中，以及找到其他页面结构元素是定位和抓取页面内容所必需的。

开发者工具的另外一个非常棒的用处是研究元素。如果右击页面的一部分，你应该会看到一个包括了一些有用的用于网页抓取的工具的菜单。图 11-4 展示了一个菜单的实例。

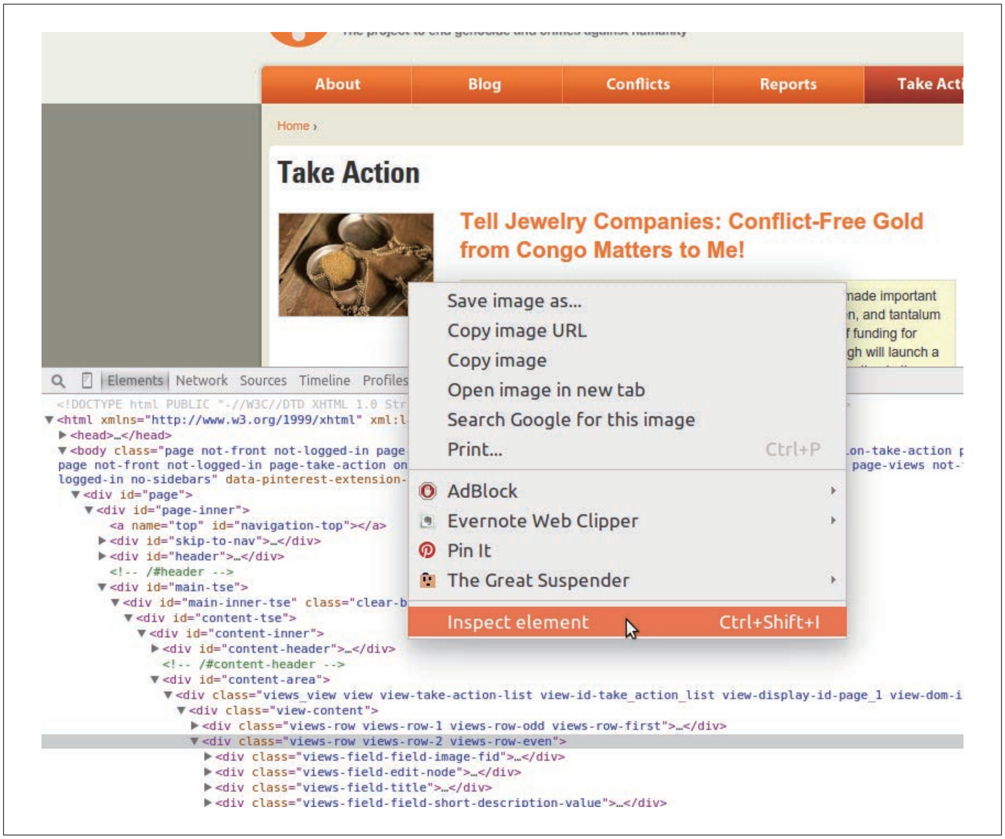


图 11-4：检视元素

如果点击“检视元素”选项，开发者工具应该会在源代码中打开那个元素。这是一个非常有用的特性，可用于与内容交互，以及查看其在代码中的位置。

除了能够在浏览器中同元素交互之外，你还可以在源代码部分同元素交互。图 11-5 展示了通过在标记结构区域右击一个元素得到的菜单类型。可以看到复制 CSS 选择器或 XPath 选择器（我们会在本章中使用这两者定位和抽取网站内容）的选项。

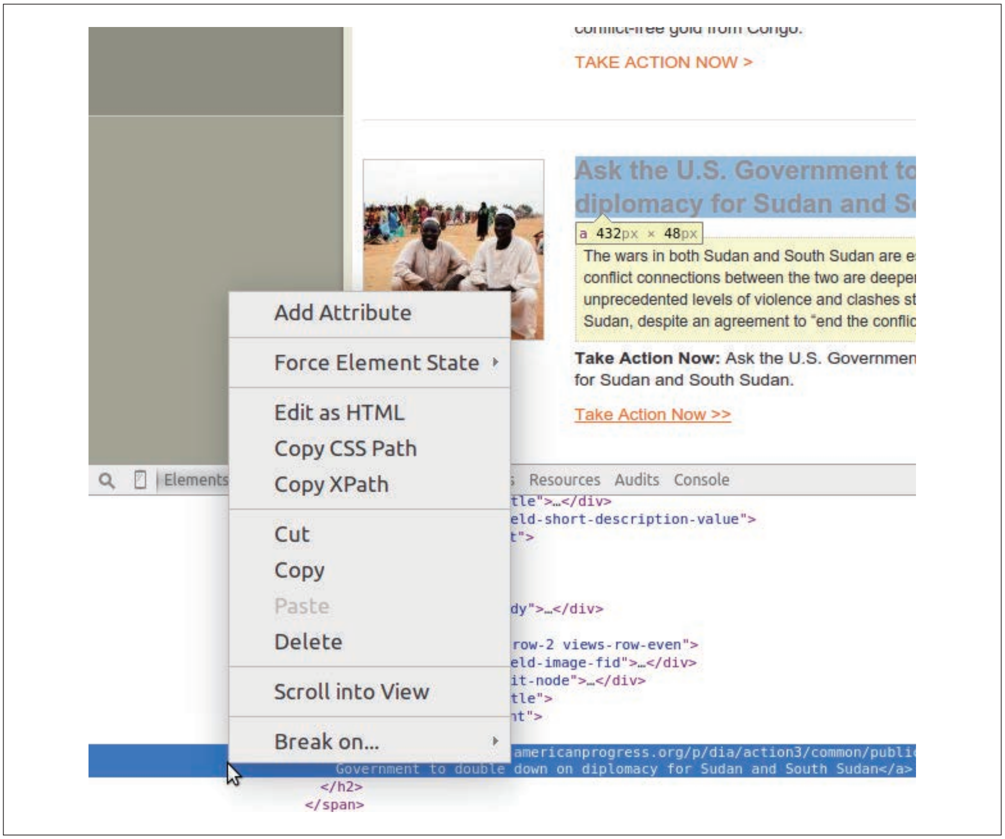


图 11-5：元素选项

> 对于不同的浏览器，工具的语言和交互可能差别很大，但是菜单选项应该同在这里描述的类似，这应该让你对如何访问数据和这些交互操作有了一些了解。

除了找到元素和内容之外，开发者工具展示了大量有关页面上节点结构和家族关系的信息。在开发者工具中，通常会有一个检视标签，展示当前元素的父元素列表。该列表中的元素通常可以被点击或者选择，所以你可以通过一个简单的点击遍历 DOM。在 Chrome 中，这个列表在开发者工具和上方页面之间的灰色部分。

我们已经查看了 Web 页面的组织结构，以及如何同它们交互来更好地理解内容的位置。现在来研究浏览器中让网页抓取更加简单的其他强大工具。

11.2.2　网络/时间线：页面是如何加载的

分析开发者工具中的时间线（Timeline）/ 网络（Network）标签将让你深刻理解页面内容是如何加载的以及加载的顺序。页面加载的时间和方式可以极大地影响你决定抓取页面的

方式。有时，理解内容来源是抓取所需内容的“快捷方式”。

网络或时间线标签展示了已加载的 URL、加载顺序和加载所需时间。图 11-6 展示了 Enough 项目页面在 Chrome 中网络标签的样子。对于不同的浏览器，你可能需要重新加载页面，来查看网络标签页面的内容。

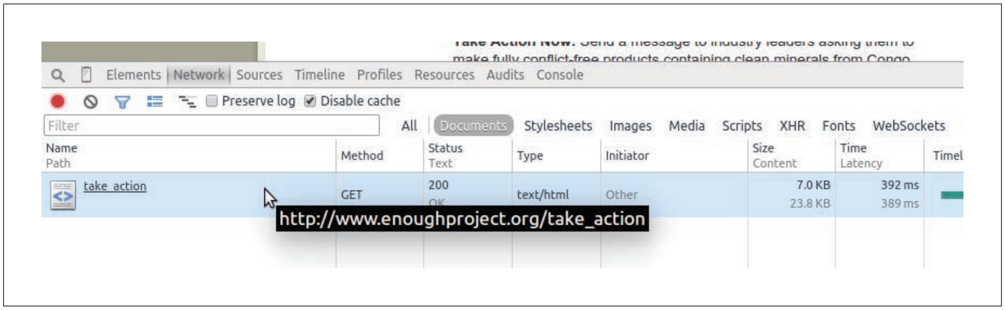


图 11-6：一个页面的网络标签

由于在网络标签中只有一个请求，可以看到整个页面在一次调用中加载完成。这对于网页抓取器来说是很棒的消息，因为这意味着通过一个请求可以获得一切。

如果点击这个请求，可以看到更多的选项，包括响应的源代码（见图 11-7）。当页面通过许多不同的请求加载时，查看每个请求的内容对于定位所需内容很重要。如果你需要额外的数据来加载站点，可以通过点击网络标签中的头部标签来研究头部和 cookie。

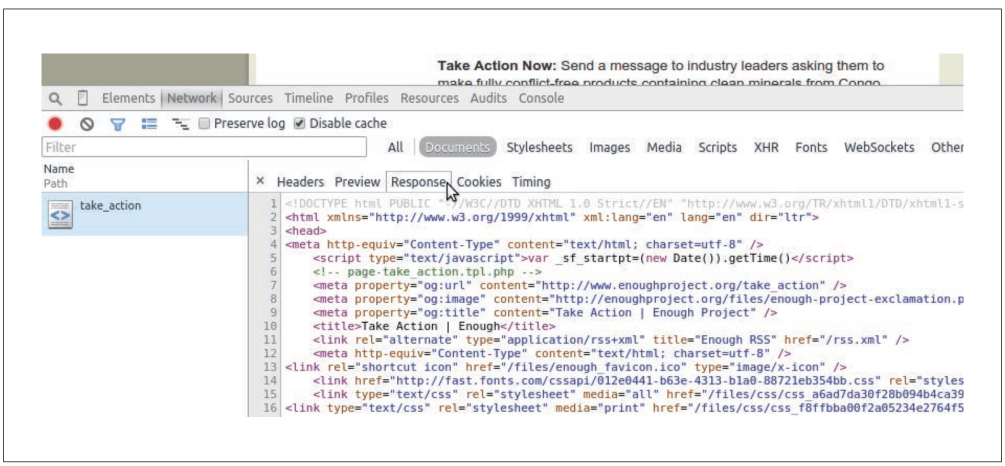


图 11-7：网络响应

来查看一个有着复杂网络标签的相似组织的页面。打开你的网络标签，使用浏览器访问 Fair phone 倡议站点上的 #WeAreFairphone 页面（http://www.fairphone.com/we-are-fairphone/，图 11-8）。

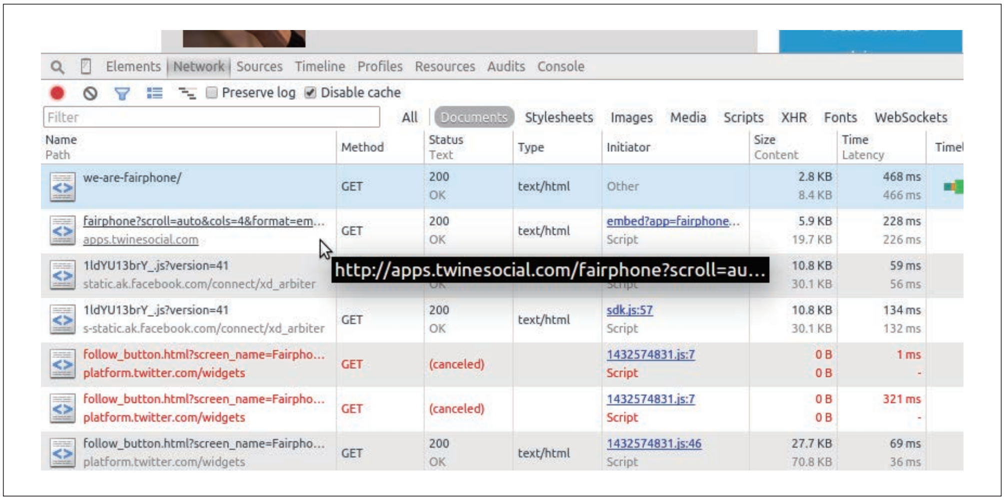


图 11-8：有很多页面的网络标签

你可以立即看到这个页面正在处理更多的请求。点击每一个请求，你可以看到每一个请求加载的内容。请求顺序显示在网络标签中的时间线上。这可以帮助你理解如何抓取和处理页面，来得到需要的内容。

通过点击每一个请求，可以看到初始页面加载后再加载大部分内容。点击初始页面的请求，会发现并没有什么内容。我们想要问的第一个问题是：这里是否有一个 JavaScript 请求或其他的请求使用 JSON 加载内容？如果有的话，对于我们的脚本来说，这可能是一个恰当的“快捷方式”。

> 你知道如何解析和读取 JSON（第 3 章），所以如果你在网络标签中找到一个 URL，伴随着一个 JSON 响应，其中保存着你需要的数据，那么你可以使用这个 URL 来获得数据，之后直接从响应中解析数据。你需要意识到所有可能在请求中需要发送的头部（展示在网络标签中的头部小节），以得到正确的响应。

如果这里没有简单的 JSON URL 匹配你需要的信息，或者信息散落在几个不同的的请求中，需要人工整合它们到一起，那么可以确定，你需要使用一个基于浏览器的方法来抓取站点。基于浏览器的网页抓取允许你读取看到的页面，而不仅是每一个请求。如果你需要在正确抓取内容之前同一个下拉菜单交互，或执行一系列基于浏览器的操作，这可以很有用。

网络标签帮助你找到包含所需内容的请求，以及是否有优秀的备选数据源。我们接下来会查看 JavaScript，看看这是否也会提供一些关于抓取器的想法。

11.2.3　控制台：同JavaScript交互

现在已经分析了页面的标记和结构，以及页面加载和网络请求的时间线，让我们转到

JavaScript 控制台，来看一下通过和运行在页面上的 JavaScript 交互，可以学到什么。

如果你已经对 JavaScript 很熟悉，使用起来应该相当简单；如果你从未同 JavaScript 交互过，花一些时间查看关于 JavaScript 课程的介绍（http://www.codecademy.com/en/tracks/javascript）会很有用。你只需要理解 JavaScript 的基本语法，能够通过控制台同页面上的元素交互。我们会从学习 JavaScript 和基本的样式开始，学习如何使用控制台界面。

1. 样式基础

每一个网页都会使用一些样式元素来帮助它组织内容、控制内容的大小和颜色，并在视觉上修改内容。当浏览器开始开发 HTML 标准，样式标准也就诞生了。样式标准的产物是级联样式表，即 CSS，这为我们提供了给页面添加样式的标准方式。举个例子，如果想要所有的标题使用不同的字体，或所有的照片在页面居中显示，你需要在 CSS 中编写这些规则。

CSS 允许样式级联，或者从父样式和样式表中继承。如果我们为整个站点定义一个样式集合，内容管理系统将很容易让每个页面看起来相似。即使我们有一个复杂站点，它有很多不同的页面类型，我们也可以定义一个主要的 CSS 文档和几个次要的文档，在页面需要额外的样式时加载次要文档。

CSS 有效，因为它定义了允许通过标签中的属性组合 DOM 元素的规则。是否记得在第 3 章研究 XML 时，讨论过嵌套属性？ CSS 同样使用这些嵌套的属性。让我们学习使用元素检视工具。因为你很可能仍然在 Fairphone 站点，所以让我们看一些页面上的 CSS 属性。当在底部工具栏高亮一个元素时，会看到一些与页面中元素相关的文本展示在旁边（图 11-9）。

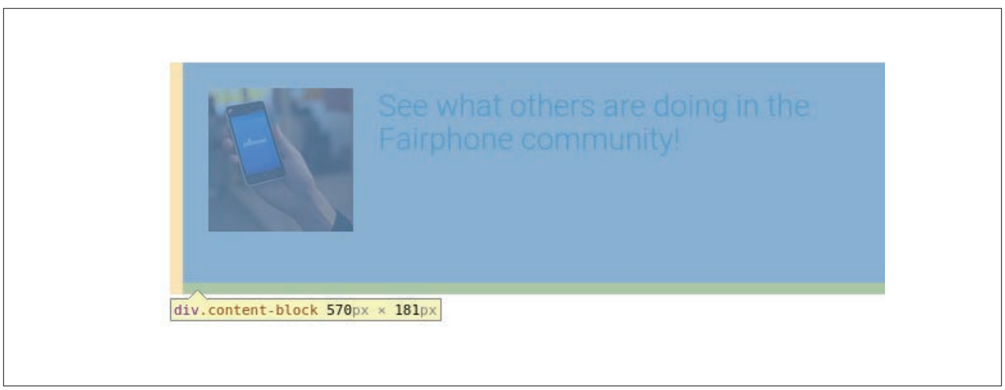


图 11-9：CSS 简介

在这个例子中，我们已经了解了 `div` 的含义，但是什么是 `contont-block` ？使用检视技术看一下 HTML 代码（右击页面上的元素，选择“检视元素”）。

我们看到 `content-block` 是 CSS 类（在图 11-10 中的嵌套属性 `class="content-block"` 中）。它定义在一个起始 `div` 标签中，同时这个 `div` 保存着所有其他子标签。说到 CSS 类，在页面的这个部分中，你能看到多少个类？有好多啊！



图 11-10：CSS 类

像类一样，同样有 CSS ID。让我们找一个（见图 11-11），看看它与类有什么不同。

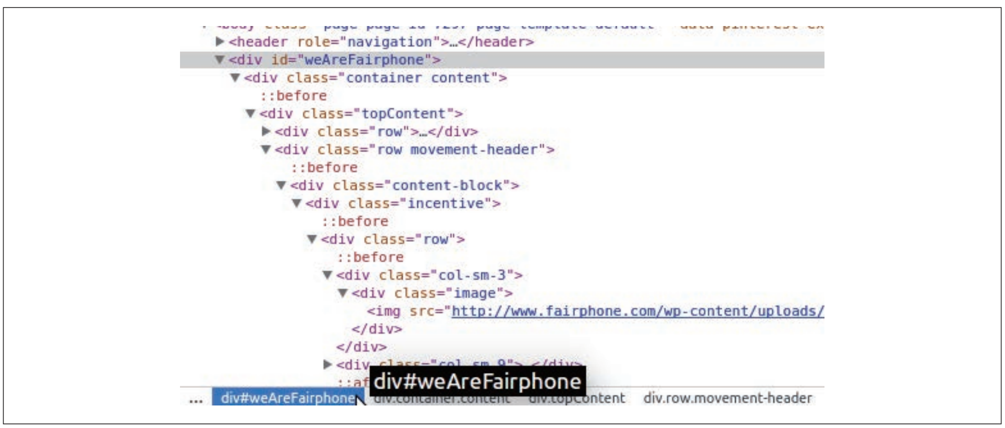
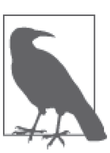


图 11-11：CSS ID

HTML 看起来很类似，但是在导航栏的符号中使用了一个哈希或英镑符号。`#` 是一个适用于 ID 的 CSS 选择器。对于类，我们使用 `.`（如同在 `div.content-block` 中展示的）。



CSS 结构和语法要求 `id` 必需是唯一的，但是你可以有很多的元素有相同的 `class`。尽管页面不总是符合这一结构，但是这也值得注意。一个 CSS `id` 相对于一个 `class` 有更大的特异性。一些元素不只有一个 `class`，所以它们可以应用多个样式。

使用右击菜单，在页面上复制 CSS 选择器相当简单。如果你已经了解了 CSS，这些知识会帮助你进行网页抓取。如果你不是太了解 CSS，但是希望更深入地探索它，可以查看 Codecademy 关于 CSS 课程的介绍（https://www.codecademy.com/courses/web-beginner-en-TlhFi/0/1）或查看 Mozilla 开发者网络的参考和指南（https://developer.mozilla.org/en-US/docs/Web/CSS）。

现在我们进一步了解了 CSS，了解了它是如何样式化页面的；但是你可能会问，在浏览器终端中 CSS 需要做什么？好问题！让我们回顾一下 jQuery 和 JavaScript 的基础知识，这样可以看到 CSS 是如何同页面上的内容交互的。

2. jQuery和JavaScript

JavaScript 和 jQuery 的演化历史要比 HTML 和 CSS 悠久得多，一部分原因是 JavaScript 的开发在很长一段时间里没有一整套的标准。从某种意义上来讲，JavaScript 是（在某种程度上仍然是）网站版图上一片荒芜的西部风光。

> 即使 JavaScript 在过去的 10 年间改变了很多，但有 10 年历史的脚本仍然经常运行在一些浏览器中，这意味着推进标准化（如何编写 JavaScript 和哪些事在 JavaScript 中不允许）的进程相对于 HTML 和 CSS 有些慢。

JavaScript 不是标记语言，而是一门脚本语言。因为 Python 也是一门脚本语言，所以你可以将一些已经学过的东西——函数、对象、类、方法——应用到对 JavaScript 的理解中去。同 Python 一样，有其他的库和包帮助你编写清晰、简单和高效的 JavaScript 代码，以便于浏览器和人们理解。

jQuery（https://jquery.com/）是一个 JavaScript 库，很多大型的网站使用它以期让 JavaScript 更易读、编写更简单，同时仍然允许浏览器（和它们不同的 JavaScript 引擎）来解析脚本。

> 早在 2005~2006 年，jQuery 引入了简化和标准化 JavaScript 的想法，为 JavaScript 开发者提供了工具，这样他们就不需要从零开始编写所有的代码。jQuery 的确推动了 JavaScript 向前发展，通过强大而易于解释的方法，以及在选择页面元素时与 CSS 更紧密的联系，创建了一个更加面向对象的方法。

自从 jQuery 被开发之后，JavaScript 和 CSS 的关系便更加紧密了，并且很多新的 JavaScript 框架都基于这个面向对象的方法。如果一个站点正在运行 jQuery，使用 CSS 标识符同页面上的元素交互很简单。假如我们想要从 #WeAreFairphone 页面（图 11-12）上正在查看的 `content-block` 类抓取内容，通过 JavaScript 控制台该如何实现呢？

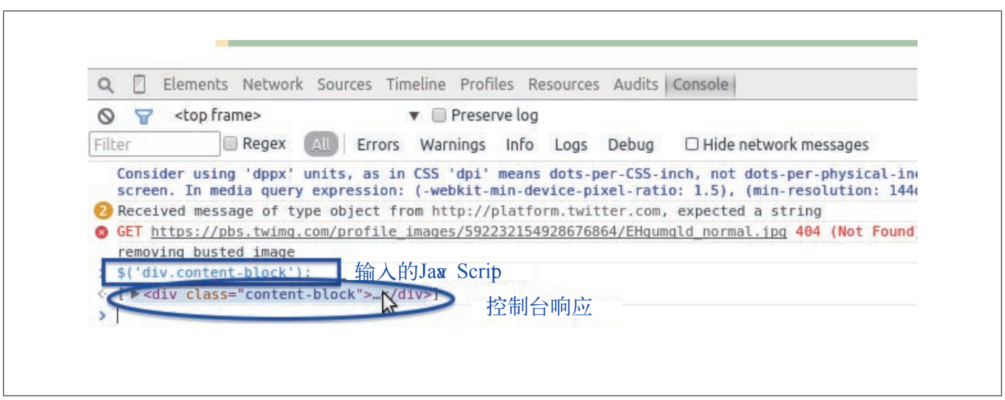


图 11-12：jQuery 控制台

由于网站正在运行 jQuery，直接在控制台标签第一行中输入下面的代码 ：

```
$('div.content-block');
```

敲击回车键，控制台会响应那个元素。点击控制台中的响应，你会看到子元素，以及该元素的子元素。可以同一些基本的 jQuery（例如，`$(elem);`）一起使用 CSS 选择器，来选择其他页面上的元素。使用 `$` 和括号会告诉 jQuery 我们正在寻找一个与括号中的字符串传递的选择器相匹配的元素。

你能使用控制台来选择 ID 为 `weAreFairphone` 的 `div` 吗？你是否只能选择页面上的锚标记（`a`）？在控制台中尝试一下。命令行和 jQuery 提供了一个简单的方式来使用 CSS 选择器或标签名称同页面上实际的元素交互，并从这些元素中拉取内容。但是这与 Python 有什么关系呢？

因为 jQuery 改变了人们对 CSS 选择器用处的看法，Python 抓取库现在使用这些选择器来遍历和寻找网页中的元素。就像你可以在浏览器控制台中使用简单的 jQuery 选择器，你也可以在 Python 抓取器代码中使用它。如果想学习更多的 jQuery 知识，建议你访问 jQuery 学习中心（https://learn.jquery.com/），或在 Codecademy（http://www.codecademy.com/en/tracks/jquery）或 Code School（https://www.codeschool.com/courses/try-jquery）上学习课程。

如果碰到一个没有使用 jQuery 的网站，那么 jQuery 在你的控制台中就不会工作。为了只使用 JavaScript 通过类来选择元素，运行：

```
document.getElementsByClassName('content-block');
```

你应该看到相同的 `div`，并且能够通过相同的方式在控制台中浏览。现在你大致知道了可以利用的工具，所以让我们更仔细地看一下如何确定抓取页面中感兴趣内容的最佳方式。首先，我们会学习如何研究页面中所有的部分。

11.2.4　页面的深入分析

一个开发 Web 抓取器的好方式是先在浏览器中分析内容。首先选择你最感兴趣的内容，并且在浏览器检视或 DOM 标签中观察。数据是如何组成的？哪里是父节点？内容包含在许多元素中，还是少量元素中？

> 在开始抓取一个页面之前，通过查看内容的限制信息和站点的 robots.txt 文件，来查看自己是否有权利抓取这个页面。你可以输入域名，随后输入 /robots.txt 找到这个文件（例如，http://oreilly.com/robots.txt）。

之后移向网络 / 时间线标签（见图 11-6）。页面的第一次加载看起来是什么样子？页面加载中是否使用了 JSON ？如果是的话，文件看起来什么样子？是否大部分内容在初次请求之后加载？所有的这些答案会帮助你确定要使用哪种类型的抓取器，以及抓取该页面有多困难。

然后，打开控制台标签。尝试使用你检视得到的信息，同包含重要内容的元素交互。对于这个内容，编写一个 jQuery 选择器有多简单？在整个域名下，你的选择器有多可靠？你是否可以打开一个类似的页面，使用该选择器，并得到类似的结果？

> 如果在 JavaScript 控制台中使用 jQuery 或 JavaScript 很容易与你的内容交互，那么使用 Python 处理可能也会很简单。如果使用 jQuery 选择一个元素很困难，或者在一个页面上可以使用的代码在另一个相似的页面上不起作用，很可能在 Python 中也同样困难。

不能使用 Python 工具正确解析的网页少之又少。我们会教给你一些技巧，来应对混乱的网页、内联 JavaScript、格式化糟糕的选择器，以及你能在万维网的代码中发现的所有糟糕的选择，同时还会给出一些最佳实践。首先，看看加载和读取网页。

11.3 得到页面：如何通过互联网发出请求

网页抓取器的第一步是……连接到互联网。让我们温习一下连接互联网的一些基础知识。

当你打开浏览器，输入一个站点名称或者搜索词，并且敲击回车键的时候，你正在发出一个*请求*。大多数情况下，这是一个 HTTP（超文本传输协议）请求（或者 HTTPS——安全版本的 HTTP 协议）。你很可能在创建一个 `GET` 请求，这是在互联网上使用的众多*请求方法之一*（https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods）。浏览器会处理这些请求，同时解析输入，以确定你是在请求一个网站还是一个搜索词。根据该分析结果，浏览器会返回搜索结果或你请求的网站。

让我们看一下用于请求 URL 的 Python 内置库：`urllib`（https://docs.python.org/2/library/urllib.html）和 `urllib2`（https://docs.python.org/2/library/urllib2.html）。这是用于 URL 请求的两个 Python 标准库。使用 `urllib2` 是个好的想法，但是在 `urllib` 中有几个很有用的方法。让我们来看一下：

```
import urllib
import urllib2

google = urllib2.urlopen('http://google.com') ❶

google = google.read() ❷

print google[:200] ❸

url = 'http://google.com?q='
url_with_query = url + urllib.quote_plus('python web scraping') ❹

web_search = urllib2.urlopen(url_with_query)
web_search = web_search.read()

print web_search[:200]
```

❶ 使用 `urlopen` 方法来开始请求。这会返回一个缓冲区，在这里你可以读取网页的内容。
❷ 读取整个页面的内容到 `google` 变量中。
❸ 打印前 200 个字符，这样可以看到页面的开端。
❹ 使用 `quote_plus` 方法来用加号转义字符串。这在处理网站的查询字符串时很有用——我们想要使用 Google 搜索网页结果，同时我们知道 Google 希望得到一个在单词之间使用加号连接的查询字符串。

看到了吗？访问 URL 或服务（例如 Google 搜索）、得到响应，并读取响应都非常简单。urllib 和 urllib2 都有其他请求方法，也能够添加头部信息、发送基本认证，以及组装更加复杂的请求。

根据请求的复杂度，你还可以使用 requests 库（http://docs.python-requests.org/en/latest/）。requests 使用 urllib 和 urllib2，让复杂的请求更容易格式化和发送。如果你需要格式化一个复杂的文件 post 请求（http://docs.python-requests.org/en/latest/user/quickstart/#more-complicated-post-requests），或查看 session（http://docs.python-requests.org/en/latest/user/quickstart/#cookies）中还存留了什么 cookie，或者检查响应状态码（http://docs.python-requests.org/en/latest/user/quickstart/#response-status-codes），requests 是一个很棒的选择。

> 在检查网络（或时间线）标签时，有些时候会发现一些使用特殊 HTTP 头（http://en.wikipedia.org/wiki/List_of_HTTP_header_fields）、cookies 或其他认证方法的页面。你可以使用 urllib2、urllib 或 requests 库，同请求一起发送这些特殊字段。

让我们看一些 requests 工具的实际使用：

```
import requests

google = requests.get('http://google.com') ❶

print google.status_code ❷

print google.content[:200]

print google.headers ❸

print google.cookies.items() ❹
```

❶ 调用 requests 库的 get 方法发送一个 GET 请求到 URL 地址。
❷ 调用 status_code 属性来确保得到了 200 响应（正确地完成请求）。如果没有得到 200，可以以不同方式执行脚本逻辑。
❸ 检查响应的 headers 属性来看 Google 返回了什么头部。可以看到 headers 属性是一个字典。
❹ 使用 cookies 属性读取 Google 在响应中发送的 cookie，并且在返回的字典上调用 items 方法来展示键 / 值对。

使用 requests 库，可以基于响应和它的属性做不同的抉择。它很容易使用，并且有很棒的文档。无论使用 urllib 还是 requests，你可以用简单的几行 Python 代码创建简单和复杂的请求。现在你了解了请求网页的基本知识，可以开始解析响应了。首先学习 Beautiful Soup（https://www.crummy.com/software/BeautifulSoup/bs4/doc/），一个简单的 Python 网页解析器。

11.4 使用Beautiful Soup读取网页

Beautiful Soup 是最流行、最简单的用于网页抓取的 Python 库之一。对于不同的需求，在

网页抓取中它可能提供了你所需的一切。它很简单、直接，并且很容易学习。让我们看一下如何使用 Beautiful Soup 解析页面。首先，使用 pip 安装这个库（使用 beautifulsoup4，因为早期的版本已经不再支持和开发了）：

```
pip install beautifulsoup4
```

让我们重新看一下在早些时候检查过的一个简单的页面，即 Enough 项目的 Take Action 页面（http://www.enoughproject.org/take_action）。我们想要看一下是否可以正确地解析所有的活动调用，并且保存它们。下面是导入页面到 Beautiful Soup 中的实例，这样我们可以开始读取它：

```
from bs4 import BeautifulSoup ❶
import requests

page = requests.get('http://www.enoughproject.org/take_action') ❷

bs = BeautifulSoup(page.content) ❸

print bs.title

print bs.find_all('a') ❹

print bs.find_all('p')
```

❶ 首先，直接从 beautifulsoup4 库导入解析器。
❷ 使用 requests 库来抓取页面上的内容，这行代码将响应（和它的内容）赋值给 page 变量。
❸ 为了开始使用 Beautiful Soup 解析，这行代码传递页面内容到 BeautifulSoup 类。可以使用 content 属性获取响应的源页面。
❹ 一旦解析了页面对象，可以使用它的属性和方法。这行代码让 Beautiful Soup 找到页面中所有的 a 标签（或链接）。

可以打开一个页面，读取响应到一个 Beautiful Soup 对象，并且使用这个对象的属性来查看标题、页面中所有的段落，以及页面上所有的链接。

我们已经学习了 HTML 中有关家族关系的知识，下面查看一下页面中的关系：

```
header_children = [c for c in bs.head.children] ❶

print header_children

navigation_bar = bs.find(id="globalNavigation") ❷

for d in navigation_bar.descendants: ❸
    print d

for s in d.previous_siblings: ❹
print s
```

❶ 使用列表生成式创建一个页面中头部的所有子元素的列表。通过将 Beautiful Soup 页面对象和 .head（调取页面的头部）以及 .children 联系在一起，可以查看所有包含在头部中的节点。如果需要的话，可以解析头部的元内容，包括页面描述。

❷ 如果使用开发者工具观察页面，你会看到导航栏使用一个 CSS 选择器 ID `globalNavigation` 定义。这行代码使用页面对象的 `find` 方法，传递一个 ID，并且定位导航栏。
❸ 使用导航栏的 `descendants` 方法遍历导航栏的后继。
❹ 到导航栏的最后一个后继，这行代码使用 `.previous_sibling` 来遍历导航元素的邻居。

家族树让我们通过 Beautiful Soup 库 `page` 类中的内置属性和方法导航。正如可以从头部和导航栏示例中看到的那样，从页面中选择一个区域，并遍历孩子、后代或邻居是很容易的。Beautiful Soup 的语法非常简单，并且将元素和它们的属性链式绑定到一起（像 `.head.children`）。对此有了了解之后，让我们专注于页面的主要部分，看一下是否可以拉取一些可能感兴趣的内容。

如果通过开发者工具观察页面，会注意到一些事情。首先，看起来每一个动作对象都位于一个 `views-row div` 中。这些 `divs` 有许多不同的类，但是它们都有一个 `views-row` 类。这是开始解析的好起点。标题位于一个 `h2` 标签中，同时链接也在该 `h2` 标签中，位于一个锚标签中。对于动作的调用位于 `views-row div` 的子 `div` 中的段落里面。现在可以使用 Beautiful Soup 解析页面。

首先，我们想要利用已掌握的 Beautiful Soup 知识，以及对页面结构和导航结构方式的理解，找到内容。下面是完成这件事的代码：

```
from bs4 import BeautifulSoup
import requests

page = requests.get('http://www.enoughproject.org/take_action')

bs = BeautifulSoup(page.content)

ta_divs = bs.find_all("div", class_="views-row") ❶

print len(ta_divs) ❷

for ta in ta_divs:
    title = ta.h2 ❸
    link = ta.a

    about = ta.find_all('p') ❹
    print title, link, about
```

❶ 使用 Beautiful Soup 找到并返回类中包含字符串 `views-row` 的所有 `divs`。
❷ 打印来检查数字是否是可以在网站上看到的故事行数，预示着正确地匹配了行数据。
❸ 遍历这些行数据，并基于页面的研究获取想要的标签。标题位于一个 `h2` 标签中，并且是行中唯一的 `h2` 标签。链接是第一个锚标签。
❹ 因为不确定在每行数据中有多少个段落标签，所以匹配所有的段落标签来得到文本。由于使用了 `.find_all` 方法，Beautiful Soup 返回一个列表，而不是第一个匹配的元素。

你应该会看到类似于下面的输出：

```
<h2><a href="https://ssl1.americanprogress.org/o/507/p/dia/action3/common/
public/?action_KEY=391">South Sudan: On August 17th, Implement "Plan B" </a></h2>  <a
href="https://ssl1.americanprogress.org/o/507/p/dia/action3/common/public/
```

```
?action_KEY=391">South Sudan: On August 17th, Implement "Plan B" </a>
[<p>During President Obama's recent trip to Africa, the international community
set a deadline of August 17 for a peace deal to be signed by South Sudan's
warring parties....]
```

这些内容可能随着站点更新而改变，但是你应该会看到一个 h2 元素，之后是一个锚（a）元素，然后是每个节点段落的列表。当下的输出是混乱的，不仅因为我们正在使用一个 print，还因为 Beautiful Soup 打印了完整的元素和它的内容。相对于完整的元素，我们更想要关注于必需的部分，也就是标题文本、链接 hrefs 和段落文本。可以使用 Beautiful Soup 来仔细地查看这部分数据：

```
all_data = []

for ta in ta_divs:
    data_dict = {}
    data_dict['title'] = ta.h2.get_text() ❶
    data_dict['link'] = ta.a.get('href') ❷
    data_dict['about'] = [p.get_text() for p in ta.find_all('p')] ❸
    all_data.append(data_dict)

print all_data
```

❶ 使用 get_text 方法抽取所有来自 HTML 元素的字符串。这样会获得标题文本。

❷ 为了得到一个元素的属性，使用 get 方法。当看到 <ahref="http://foo.com">Foo，并想提取链接时，可以调用 .get("href") 来返回 href 值（即，foo.com）。

❸ 为了抽取段落文本，使用 get_text 方法，遍历 find_all 方法返回的段落。这行代码使用列表生成式来编译一个有着动作内容调用的字符串列表。

现在数据和输出呈现了一个更加有组织的格式。在变量 all_data 中，保存了一个所有数据的列表。现在每一个数据输入都和匹配键保存在其字典中。我们用一种整洁的方式，使用一些新的方法（get 和 get_text）从页面抓取了数据，并且数据现在存放在数据字典中。代码更加清晰和精确，可以通过添加辅助函数让它更加清晰（像第 8 章介绍的）。

除此之外，可以自动化脚本来检查是否有新的动作调用。如果保存数据到 SQLite，并且将其用于每月检查刚果的劳工实践，可以自动化报告。在每一个新报告中，可以抽取这些数据，并且对对抗冲突矿产和童工激起更多的兴趣。

Beautiful Soup 是一个易于使用的工具，并且其文档（https://www.crummy.com/software/BeautifulSoup/bs4/doc/）中介绍了很多其他可用方法的实例。这个库对于初学者来说很棒，并且有很多简单的函数；然而，跟一些其他的 Python 库相比，它太简单了。

由于 Beautiful Soup 的解析是基于正则表达式的，用在缺乏正确标签结构的破损网页上很有效。但是如果想要遍历更加复杂的页面，或者想要抓取器运行得更快并且快速地浏览页面，有很多更加高级的 Python 库可用。让我们看一下许多天才网页抓取器开发者最爱的库：lxml。

11.5　使用lxml读取网页

一个更高级的网页抓取器（其他的高级工具把它作为解析器来使用）是 lxml（http://lxml.

de/)。它非常强大而快速，而且有很多很棒的特性，包括生成 HTML 和 XML 以及清洗编写糟糕的网页的能力。除此之外，它有很多用于遍历 DOM 和网页家族关系的工具。

> **安装 lxml**
>
> lxml 有许多不同的 C 依赖，这使得安装它要比安装大多数 Python 库更复杂一点（http://lxml.de/installation.html）。对于 Windows 用户，可查看开源二进制构建版本的 lxml（http://lxml.de/FAQ.html#where-are-the-binary-builds）。对于 Mac 用户，建议安装 Homebrew（http://brew.sh/），这样你可以使用 brew install lxml 安装它。有关高级安装的更多细节，请查看附录 D。

让我们快速地看一下要使用的主要 lxml 特性，先重写 Beautiful Soup 的代码来使用 lxml：

```
from lxml import html

page = html.parse('http://www.enoughproject.org/take_action') ❶
root = page.getroot() ❷

ta_divs = root.cssselect('div.views-row') ❸

print ta_divs

all_data = []

for ta in ta_divs:
    data_dict = {}
    title = ta.cssselect('h2')[0] ❹
    data_dict['title'] = title.text_content() ❺
    data_dict['link'] = title.find('a').get('href') ❻
    data_dict['about'] = [p.text_content() for p in ta.cssselect('p')] ❼
    all_data.append(data_dict)

print all_data
```

❶ 这里使用 lxml 的解析方法，它可以从一个文件名、一个打开的缓冲区或一个合法的 URL 解析。它返回一个 etree 对象。

❷ 因为 etree 对象的方法和属性比 HTML 元素对象少很多，所以这行代码访问根（页面和 HTML 的顶部）元素。根包含所有可能的能够访问的主干（孩子）和细枝（后代）。从根可以向下解析每一个链接或者段落，并且可以返回整个页面的 head 和 body 标签。

❸ 使用根元素，这行代码找到所有的类名称为 views-row 的 div。它使用 cssselect 方法和一个 CSS 选择器字符串，返回一个匹配元素的列表。

❹ 为了抓取标题，使用 cssselect 方法找到 h2 标签。这行代码选择了列表中的第一个元素。cssselect 返回一个所有匹配项的列表，但是我们只想要第一个匹配的元素。

❺ 同 Beautiful Soup 的 get_text 方法类似，text_content 为 lxml HTML 元素对象返回标签（和任何子标签）内的文本。

❻ 这里使用链式方法来从 title 元素中获得锚标签，并且拉取锚标签中的 href 属性。这只返回这一属性的值，类似于 Beautiful Soup 的 get 方法。

❼ 使用列表生成式来从 Take Action div 中的每一个段落中拉取文本，组成完整的文本。

你应该看到与我们使用 Beautiful Soup 时相同的提取数据。不同的是语法和页面加载的方式。Beautiful Soup 使用正则表达式把文档作为一个长字符串解析。lxml 使用 Python 和 C 库来识别页面结构，并且用更加面向对象的方式遍历它。lxml 查看所有标签的结构，（取决于你的计算机和安装它方式的不同）使用最快的方法解析树，并且在一个 etree 对象中返回数据。

我们可以使用 etree 对象本身，或者调用 getroot，这个函数会返回树最顶部的元素——通常为 html。有了这个元素，可以使用很多不同的方法和属性读取和解析页面剩余的部分。我们的解决方案强调了一点：使用 cssselect 方法。这个方法使用 CSS 选择器字符串（类似于 jQuery 示例），并且使用这些字符串来识别 DOM 元素。

lxml 也有 find 和 findall 方法。find 和 cssselect 之间有什么主要区别呢？来看一些示例：

```
print root.find('div') ❶

print root.find('head')

print root.find('head').findall('script') ❷

print root.cssselect('div') ❸

print root.cssselect('head script') ❹
```

❶ 在根元素上使用 find 方法来找到 div，这返回空。从浏览器的检视来看，我们知道页面充满了 divs！
❷ 使用 find 方法查看头部标签，使用 findall 方法在头部定位脚本元素。
❸ 使用 cssselect 取代 find 正确地定位文档中所有的 divs，它们作为一个大的列表返回。
❹ 使用 cssselect，通过嵌套 CSS 选择器在头部定位脚本标签。使用 head script 返回与从根对象链式调用 find 命令相同的列表。

所以，find 和 cssselect 的操作方式有很大的不同。find 利用 DOM 来遍历元素，并基于祖先和家族关系找到它们，而 cssselect 方法利用 CSS 选择器来寻找页面中所有可能的匹配，或者元素的后继，非常类似于 jQuery。

> 根据需求的不同，find 或 cssselect 可能更加有用。如果页面的 CSS 类、ID 和其他标识符组织得良好，cssselect 是一个非常棒的选择。但是如果页面没有组织或不使用这些标识符，遍历 DOM 可以帮助你通过家族关系确定内容。

我们想要探索其他有用的 lxml 方法。作为一名开发者，随着不断学习和成长，你可能想要通过 emoji 表情表达进程。出于这个原因，让我们编写一个快速的 emoji 图表解析器（http://www.emoji-cheat-sheet.com/）来保存一个最新的 emoji 表情列表，你可以在 Basecamp、GitHub 和很多其他的技术相关网站上使用它们。下面是做这件事的代码：

```
from lxml import html
import requests

resp = requests.get('http://www.emoji-cheat-sheet.com/')
page = html.document_fromstring(resp.content) ❶
```

```
body = page.find('body')
top_header = body.find('h2') ❷

print top_header.text

headers_and_lists = [sib for sib in top_header.itersiblings()] ❸

print headers_and_lists

proper_headers_and_lists = [s for s in top_header.itersiblings() if
                            s.tag in ['ul', 'h2', 'h3']] ❹

print proper_headers_and_lists
```

❶ 这段代码使用 requests 库拉取 HTML 文档的主体，之后使用 html 模块的 document_fromstring 方法解析数据为一个 HTML 元素。

❷ 通过查看页面结构，可以看到这是一系列头部的匹配列表。这行代码定位第一个头部，这样我们可以使用家族关系来寻找其他有用的部分。

❸ 这行代码使用列表生成式和 itersiblings 方法（返回一个迭代器）来查看所有的邻居。

❹ 上一个 print 展示了初始的 itersibling 列表生成式返回了远超我们需求的数据，包括一些页面下方带有 div 和 script 元素的部分。使用页面检视，我们确定想要的标签只是 ul、h2 和 h3。这行代码使用列表生成式和一个 if 确保只返回目标内容。

itersiblings 方法和 tag 属性帮助我们轻松地定位想要选择和解析的内容。在这个例子中，我们没有使用任何 CSS 选择器。我们知道，代码不会因为添加一个新部分而损坏，只要页面继续在头部和列表标签中保存内容。

为什么只想使用 HTML 元素构建一个解析器呢？不依赖于 CSS 类的优势是什么？如果一个站点的开发者改变了它的设计或让它变得对移动端更友好，那么很可能他会修改 CSS 和 JavaScript，而不是重新编写页面结构。如果使用基本的页面结构驱动抓取器，它们可能会比那些使用 CSS 的抓取器用得更久，有效期更长。

除了 itersiblings 之外，lxml 对象可以迭代孩子、后继和祖先。使用这些方法遍历 DOM，是熟悉页面组织方式和编写持久代码的很好的方式。你同样可以使用家族关系来编写有意义的 XPath——一种结构化的模式，用于基于 XML 的文档（像 HTML）。尽管 XPath 不是解析网页最简单的方式，但它是一种快速、高效且极度简单的方式。

一个XPath案例

虽然使用 CSS 选择器是一种找到页面上元素和内容的简单方式，也建议你学习和使用 XPath（https://en.wikipedia.org/wiki/XPath）。XPath 是一个标记模式选择器，组合了 CSS 选择器和遍历 DOM 的能力。理解 XPath 是学习网页抓取和网站结构的很好的方式。有了 XPath，你可以访问仅仅使用 CSS 选择器不容易阅读的内容。

XPath 可以用于几乎所有主要的网页抓取库，并且比其他大多数识别和同页面内容交互的方法都快得多。事实上，大多数同页面交互的选择器方法都在库内部转化为 XPath。

为了练习 XPath，你只需要查看浏览器的工具。许多浏览器都能够查看和复制 DOM 中的 XPath 元素。微软也有一篇关于 XPath 的很棒的文章（https://msdn.microsoft.com/en-us/library/ms256086(v=vs.110).aspx），而且 Mozilla 开发者网络上有很多很棒的工具和示例（https://developer.mozilla.org/en-US/docs/Web/XPath），供你更深入地学习 XPath。

XPath 遵循特定的语法来定义元素的类型、在 DOM 中的位置，以及可能拥有什么属性。表 11-2 回顾了可以在网页抓取代码中使用的一些 XPath 语法模式。

表11-2：XPath语法

表达式	描述	示例
//*node_name*	在文档中选择所有匹配 *node_name* 的节点	`//div`（选择文档中的所有 `div` 对象）
/*node_name*	选择当前或前序元素中所有匹配 *node_name* 的节点	`//div/ul`（选择所有 `div` 内的 `ul` 对象）
@*attr*	选择一个元素的属性	`//div/ul/@class`（选择所有 `div` 中 `ul` 对象的 `class` 属性）
`../`	选择父元素	`//ul/../`（选择所有 `ul` 元素的父元素）
[@*attr*="*attr_value*"]	选择有特定属性值的元素	`//div[@id="mylists"]`（选择 ID 值为“mylists”的 `div`）
`text()`	从节点或元素中选择文本	`//div[@id="mylists"]/ul/li/text()`（选择 ID 为“mylists”的 `div` 中的列表中元素的文本）
`contains(@attr, "value")`	选择属性具有特定值的元素	`//div[contains(@id, "list")]`（选择所有 ID 中有“list”的 `div`）
`*`	通配符	`//div/ul/li/*`（选择所有的 `div` 中 `ul` 中列表对象的后继）
`[1,2,3...]`、`[last()]` 或 `[first()]`	根据在节点中出现的顺序选择元素	`//div/ul/li[3]`（选择所有 `div` 中 `ul` 中的第三个列表对象）

还有更多的表达式，但是这些已经足够我们开始了。让我们使用 XPath 和本章早些时候创建的非常漂亮的 HTML 页面，研究如何解析 HTML 元素间的家族关系。为了跟随我们，从本书的代码仓库（https://github.com/jackiekazil/data-wrangling）中将其拉取到你的浏览器中（文件：awesome_page.html）。

假设我们想要在页脚部分选择链接。通过使用“检视元素”选项（见图 11-13），可以看到底部栏展示了一个元素列表和它们的祖先。锚链接位于 `html` 标签内的 `body` 标签内的 `footer` 内的一个带有 CSS `id` 的 `div` 内的 `ul` 内的 `li` 标签里（喔！我觉得快要喘不过来气了！）。

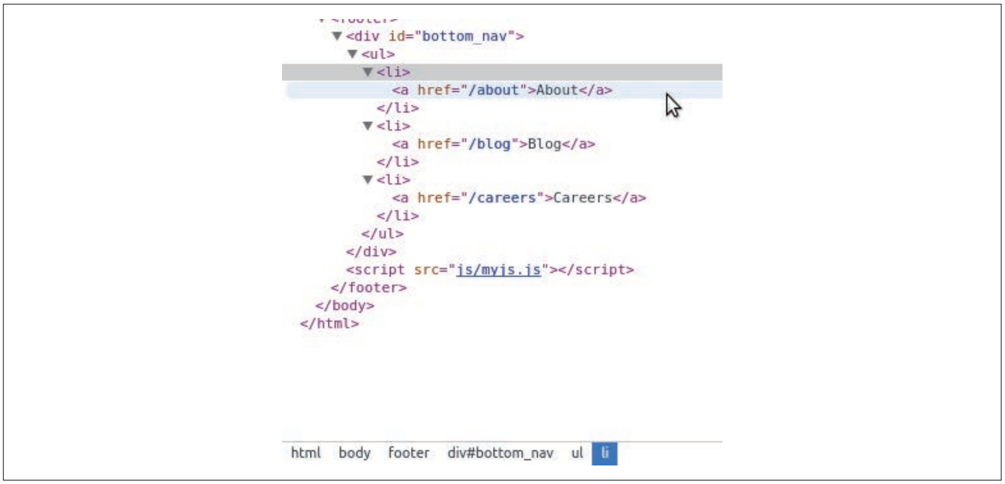


图 11-13：找到页面的元素

怎样编写 XPath 来选择它呢？实际上，有很多种方式。让我们从一个相当明显的方式开始，使用带有 CSS `id` 的 `div` 来编写 XPath。用已学到的语法选择 `div`：

```
'//div[@id="bottom_nav"]'
```

可以使用浏览器的 JavaScript 控制台测试这段代码。为了在控制台中测试 XPath，直接将它放在 `$x();` 中，这是一个 jQuery 控制台的实现，用于使用 XPath 浏览页面。让我们在控制台中查看一下（见图 11-14）。[1]

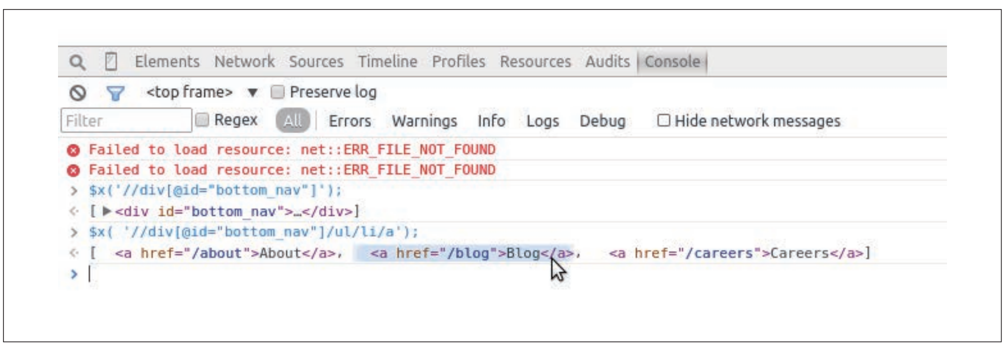


图 11-14：使用控制台编写 XPath

我们有了合法的 XPath 来选择导航栏，因为控制台返回了一个对象（类似于 jQuery 选择器）。但是我们真正想要的是链接。让我们来看一下怎样从这个 `div` 移到这些链接。我们知道它们是后继，所以编写一个家族关系。

注 1：如果你想要在一个不使用 jQuery 的站点上使用 XPath，需要使用 Mozilla 在文档中描述的不同语法（https://developer.mozilla.org/en-US/docs/Introduction_to_using_XPath_in_JavaScript）。对于这个元素，语法应该是 `document.evaluate('//div[@id="bottom_nav"]', document)`。

```
'//div[@id="bottom_nav"]/ul/li/a'
```

这里我们想要任何具有 id bottom_nav 的 divs，其中包含一个无序列表，然后是匹配项中的列表对象，再然后是这些对象中的锚标签。让我们尝试在控制台中运行它（图 11-15）。

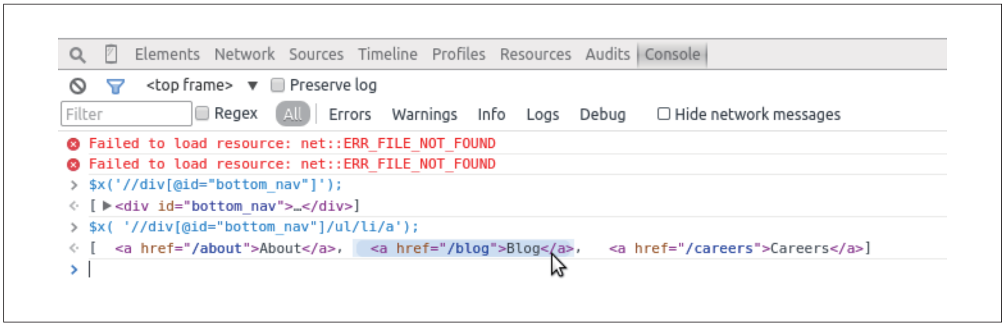


图 11-15：XPath 子元素

可以从控制台的输出看到已经选择了这三个链接。现在，我们只想提取网页地址本身。我们知道每一个锚标签有一个 href 属性。让我们使用 XPath 来为这些属性编写一个选择器：

```
'//div[@id="bottom_nav"]/ul/li/a/@href'
```

当在控制台中运行这个选择器时，可以看到我们已经正确地选择了底部链接中的网页地址（见图 11-16）。

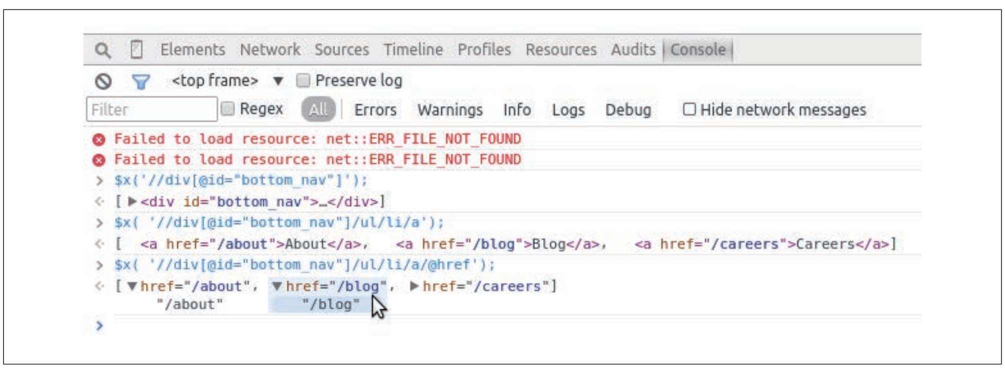


图 11-16：寻找 XPath 属性

了解页面结构可以帮助我们得到很难访问的内容，我们可以使用 XPath 表达式取而代之。

> 由于 XPath 的能力和速度，你需要慢慢学习。比如，如果在类与 ID 之间存在空间，应该使用 contains 模式，而不是 =。元素可以拥有多个类，而 XPath 会假定包含了整个类字符串；使用 contains 将帮助你找到任何包含这个子串的元素。

找到你感兴趣的元素的父元素可能会很有用。假如你对页面上的一个对象列表感兴趣，并

且你可以使用 CSS 类或列表中包含的文本轻松地定位一个或多个列表对象。你可以使用这些信息来构建一个 XPath 选择器，定位该元素，之后寻找父元素，让你能够访问整个列表。12.2.1 节探索这些 XPath 选择器类型，因为 Scrapy 利用 XPath 进行快速解析。

使用 XPath 的一个原因是，你通过 CSS 选择器找到的 CSS 类可能并不总能正确地选择元素，特别是使用了不同的驱动处理页面时（例如，Selenium 和许多浏览器）。XPath 天生更加，因而是正确解析网页的一种更加可靠的方式。

如果你已经抓取了一个站点很长时间，并且想要复用相同的代码，XPath 不太可能会由于小段代码的改变和站点的开发而崩溃。更常见的作法是重写一些 CSS 类或样式，而不是修改整个站点和页面结构。因此，XPath 比使用 CSS 更安全（尽管不是万无一失）。

现在你已经学习了一些 XPath 知识，可以尝试使用 XPath 语法重新编写 emoji 处理器，正确地存储每个部分中所有的 emoji 和头部信息。代码类似下面这样。

```
from lxml import html

page = html.parse('http://www.emoji-cheat-sheet.com/')

proper_headers = page.xpath('//h2|//h3') ❶
proper_lists = page.xpath('//ul') ❷

all_emoji = []

for header, list_cont in zip(proper_headers, proper_lists): ❸
    section = header.text
    for li in list_cont.getchildren(): ❹
        emoji_dict = {}
        spans = li.xpath('div/span') ❺
        if len(spans):
            link = spans[0].get('data-src') ❻
            if link:
                emoji_dict['emoji_link'] = li.base_url + link ❼
            else:
                emoji_dict['emoji_link'] = None
            emoji_dict['emoji_handle'] = spans[1].text_content() ❽
         else:
             emoji_dict['emoji_link'] = None
             emoji_dict['emoji_handle'] = li.xpath('div')[0].text_content() ❾
         emoji_dict['section'] = section
         all_emoji.append(emoji_dict)

print all_emoji
```

❶ 这行代码寻找与 emoji 内容相关的头部信息。它使用 XPath 抓取所有的 `h2` 和 `h3` 元素。
❷ 每一个定位到的头部有一个 `ul` 元素来匹配。这行代码在整个文档中收集所有的 `ul` 元素。
❸ 使用 `zip` 方法来打包头部和与之适合的列表，这返回一个元组列表。这行代码之后解包这些元组，使用一个 `for` 循环拉取每一个部分（头部与列表内容）到独立的变量中。
❹ 这段代码遍历 `ul` 元素的子元素（`li` 元素保存着 emoji 表情信息）。
❺ 通过页面检视，我们知道大多数的 `li` 元素有一个 `div`，其中包含两个 `span` 元素。这些 `span` 包括 emoji 表情的图片链接，以及用来唤起 emoji 表情的文字。这行代码使用

XPath 的 `div/span` 返回每个子 `div` 元素下所有的 `span` 元素。

❻ 为了找到每个元素的链接，这行代码调用第一个 `span` 的 `data-src` 属性。如果 `link` 变量为 `None`，代码会在我们的数据字典中设置 `emoji_link` 属性为 `None`。

❼ 因为 `data-src` 保存着一个相对 URL，所以这行代码使用 `base_url` 属性来创建一个完整的绝对 URL。

❽ 为了得到句柄（handle）或唤起 emoji 表情所需的文字，这行代码抓取第二个 `span` 的文本。不同于链接的逻辑，我们不需要测试这是否存在，因为每一个 emoji 都拥有一个句柄。

❾ 对于包括 Basecamp 声效的页面，对于每一个列表对象，存在一个 `div`（你可以通过使用浏览器的开发者工具检视页面，轻松地找到它）。这行代码选择 `div`，并且抓取其中的文本内容。因为这行代码在 `else` 代码块中，所以我们知道这些只是声音文件，因为它们不使用 `spans`。

通过重写 emoji 代码来使用 XPath 关系，我们发现标签最后的代码块是声音，并且其中的数据以不同的方式存储。相对于在 `span` 中保存一个链接，这里只有一个 `div` 包含唤醒声音的文本。如果只想要 emoji 链接，可以跳过添加它们到列表对象的迭代。取决于你感兴趣的数据，代码会有很大相同，但是你总是可以轻松地利用 `if...else` 逻辑来确定需要的内容。

通过不超过 30 行的代码，我们创建了一个抓取器来请求页面，通过 XPath 遍历 DOM 关系解析它，同时使用合适的属性或文本内容抓取出需要的内容。这段代码具有很好的扩展性，如果页面的作者添加了更多的数据节，只要页面结构没有大幅度改变，解析器会继续从页面拉取内容，并且我们会拿到不计其数的 emoji 表情！

还有许多其他有用的 `lxml` 函数。表 11-3 总结了其中一些以及它们的使用场景。

表11-3：`lxml`特性

方法或属性名称	描述	文档
`clean_html`	一个用来清理糟糕格式页面的函数，这样它们可以被正确解析	http://lxml.de/lxmlhtml.html#cleaning-up-html
`iterlinks`	一个用来访问页面上每一个锚标签的迭代器	http://lxml.de/lxmlhtml.html#working-with-links
`[x.tag for x in root]`	所有的 `etree` 元素可以作为简单的迭代器使用，支持子元素的遍历	http://lxml.de/api.html#iteration
`.nsmap`	提供对命名空间的简单访问，如果你愿意使用它们的话	http://lxml.de/tutorial.html#namespaces

现在，当研究页面上的结构化数据和解决如何使用 `lxml`、Beautiful Soup 和 XPath 从页面中提取内容时，你应该感到很自信。下一章会继续研究其他可以用来做不同类型抓取的库，像基于浏览器的解析和爬虫。

11.6　小结

你已经学习了许多关于网页抓取的知识。在编写不同格式的抓取器时，你应该感到很自信。你已清楚怎样编写 jQuery、CSS 和 XPath 选择器，以及如何轻松地使用浏览器和 Python 匹配内容。

在使用开发者工具分析一个网页是如何构建的时候，你同样会感到很自在。你已经磨练了 CSS 和 JavaScript 技能，学习了如何编写一个合法的 XPath 来与 DOM 树直接交互。

表 11-4 列出了本章介绍的新概念和库。

表11-4：新的Python和编程概念与库

概念/库	目的
robots.txt 文件使用、版权和商标研究	通过站点的 robots.txt 文件、服务条款或页面上发布的其他法律声明，你可以确定是否可以合法和符合道德地抓取站点内容
开发者工具使用：检视 /DOM	用于研究内容在页面上的位置，以及如何以最佳方式使用页面层次和 CSS 规则来找到它
开发者工具使用：网络	用于研究为了完全加载页面发起了哪些调用。这其中的一些请求可能指向 API，或其他资源，以便你轻松获取数据。了解页面如何加载可以帮助你确定是使用一个简单的抓取器还是一个基于浏览器的更复杂的抓取器
开发者工具使用：JavaScript 控制台	用于研究如何通过其 CSS 或 XPath 选择器同页面上的元素交互
`urllib` 和 `urllib2` 标准库	帮助你创建简单的 HTTP 请求来访问一个网页，并通过 Python 标准库获取内容
`requests` 库	帮助你更容易地创建复杂的页面请求，特别是那些需要额外的头部、复杂的 `POST` 数据或请求认证
`BeautifulSoup` 库	让你轻松读取和解析页面。对于严重破损的页面和初始的网页抓取很有用
`lxml` 库	让你更轻松地使用类似 XPath 语法的 DOM 层次结构和工具解析页面
XPath 使用	使你能够使用正则表达式和 XPath 语法编写模式和匹配，快速地找到和解析页面内容

在下一章，你会学习更多从网页抓取数据的方式。

第 12 章

高级网页抓取：屏幕抓取器与爬虫

在第 11 章你已经开始培养网页抓取技能，学习了如何确定要抓取的内容，以及用什么方式去哪里抓取。在这一章，我们会学习用更高级的抓取器来收集内容，比如基于浏览器的抓取器和爬虫。

我们还会学习使用高级网页抓取工具调试常见问题，并介绍在抓取网页时会遇到的一些道德问题。首先，我们会研究基于浏览器的网页抓取：通过 Python 直接使用浏览器从网页上抓取内容。

12.1 基于浏览器的解析

有时，站点使用大量的 JavaScript 或其他页面加载后执行的代码来给页面填充内容。在这些情况中，使用一个普通的网页抓取器来分析站点几乎是不可能的。你最后得到的是一个空白的页面。如果你想要同页面进行交互（即，如果你需要点击按钮或者输入一些搜索文本），也会碰到相同的问题。无论哪一种情况，你需要找出屏幕阅读（screen read）页面的方法。屏幕读取器使用浏览器打开页面，在浏览器中加载页面之后读取并同它交互。

> 屏幕读取器很擅长执行通过一系列操作来获取信息才能完成的任务。出于这个原因，屏幕读取器脚本也是自动化常规网页任务的简单方式。

在 Python 中最常用的屏幕读取库是 Selenium（http://selenium.googlecode.com/svn/trunk/docs/api/py/index.html）。Selenium 是一个 Java 程序，用来打开浏览器，并且通过读取页面同页面交互。如果你已经了解 Java，可以使用 Java IDE 来与浏览器交互。我们会通过 Python 使用 Python 包（Python binding）与 Selenium 交互。

12.1.1 使用Selenium进行屏幕读取

Selenium 是一个强大的基于 Java 的引擎，可通过支持 Selenium 的浏览器直接与网站交互。它是一个非常流行的用于用户测试的框架，让公司可以为它们的网站构建测试。对于我们的目标，我们会使用 Selenium 来抓取一个我们需要交互或不是所有内容都在第一次请求中加载（例如图 11-6 中的示例，大多数的内容在第一次请求完成后加载）的站点。让我们查看页面，看看是否可以通过 Selenium 读取它。

首先，需要使用 `pip install` 安装 Selenium（http://selenium-python.readthedocs.io/en/latest/installation.html）：

```
pip install selenium
```

现在，开始编写 Selenium 代码。首先，需要打开浏览器。Selenium 支持许多不同的浏览器，但是附带了一个 Firefox 的内置驱动。如果你没有安装 Firefox，可以安装它，或者为 Chrome（https://code.google.com/p/selenium/wiki/ChromeDriver）、IE（https://code.google.com/p/selenium/wiki/InternetExplorerDriver）或 Safari（https://code.google.com/p/selenium/wiki/SafariDriver）安装 Selenium 驱动。让我们看一下是否能够使用 Selenium 打开网页（在例子中，我们会使用 Firefox，但是切换和使用不同的驱动器是很简单的）：

```
from selenium import webdriver ❶

browser = webdriver.Firefox() ❷
browser.get('http://www.fairphone.com/we-are-fairphone/') ❸

browser.maximize_window() ❹
```

❶ 导入来自 Selenium 的 `webdriver` 模块。这个模块用来调用任何已经安装的驱动器。

❷ 通过使用 `webdriver` 模块的 `Firefox` 类初始化 Firefox 浏览器对象。这会在计算机上打开一个新的浏览器窗口。

❸ 通过 `get` 方法和一个 URL 参数，访问想要抓取的 URL。打开的浏览器应该开始加载页面了。

❹ 使用 `maximize_browser` 方法最大化打开的浏览器。这会帮助 Selenium“看到”更多的内容。

现在已经有了一个页面加载完的浏览器对象（`browser` 变量）。让我们看一下是否能够同页面上的元素交互。如果你使用浏览器的检视标签，会看到社交媒介内容在一个类名称为 `content` 的 `div` 中。让我们看一下是否可以使用新的 `browser` 对象看到全部内容：

```
content = browser.find_element_by_css_selector('div.content') ❶

print content.text ❷

all_bubbles = browser.find_elements_by_css_selector('div.content') ❸

print len(all_bubbles)

for bubble in all_bubbles: ❹
    print bubble.text
```

❶ browser 对象有一个函数 find_element_by_css_selector，使用 CSS 选择器来选择 HTML 对象。这行代码选择了第一个类名称为 content 的 div，这会返回第一个匹配的对象（一个 HTMLElement 对象）。
❷ 这行代码会打印第一个匹配对象的文本内容。我们期待看到第一个聊天气泡。
❸ 这行代码使用 find_elements_by_css_selector 方法，传递一个 CSS 选择器，找到所有匹配的对象。这个方法返回一个 HTMLElement 对象列表。
❹ 遍历列表，并且打印每一个对象的内容。

嗯，有些奇怪。看起来只有两个匹配我们想要查找的对象（因为在打印 all_bubbles 的长度时，看到输出为 2），但是我们在页面上看到了大量的内容气泡。让我们更深入地查看页面上的 HTML 对象，看是否可以弄明白为什么没有匹配出更多的对象（见图 12-1）。



图 12-1：iframe

啊！当查看内容的父元素时，我们看到这是一个 iframe（https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe），位于页面的中部。iframe（内联框架）是一个 HTML 标签，它将另外一个 DOM 结构嵌入到页面中，允许一个页面在它自身中加载另外一个页面。我们的代码可能不能解析它，因为解析器希望只遍历一个 DOM 对象。让我们看一下是否可以让 iframe 在一个新的窗口中加载，这样就不需要遍历两个 DOM。

```
iframe = browser.find_element_by_xpath('//iframe') ❶

new_url = iframe.get_attribute('src') ❷

browser.get(new_url) ❸
```

❶ 使用 find_element_by_xpath 方法，返回第一个匹配 iframe 标签的元素。
❷ 得到 src 属性，这包含在 iframe 中加载的页面的 URL。
❸ 在浏览器中加载 iframe 的 URL。

我们找到了如何加载想要的内容的方法。现在看一下是否可以加载所有的内容气泡：

```
all_bubbles = browser.find_elements_by_css_selector('div.content')

for elem in all_bubbles:
    print elem.text
```

现在有了气泡内容——很棒！让我们收集一些信息：想要提取人的姓名、他们分享的内容、照片（如果有的话），以及到原始内容的链接。

在浏览页面 HTML 代码的过程中，看起来每一个内容元素都有 `fullname` 和 `name` 元素来标识个人，还有一个有文本内容的 `twine-description` 元素。我们看到这里有一个 `picture` 元素，还有一个 `when` 元素，其中保存着时间数据。`when` 元素还包含一个原始链接。具体讲解如下。

```
from selenium.common.exceptions import NoSuchElementException ❶

all_data = []

for elem in all_bubbles: ❷
    elem_dict = {}

    elem_dict['full_name'] = \
        elem.find_element_by_css_selector('div.fullname').text ❸
    elem_dict['short_name'] = \
        elem.find_element_by_css_selector('div.name').text
    elem_dict['text_content'] = \
        elem.find_element_by_css_selector('div.twine-description').text
    elem_dict['timestamp'] = elem.find_element_by_css_selector('div.when').text
    elem_dict['original_link'] = \
    elem.find_element_by_css_selector('div.when a').get_attribute('href') ❹
try:
    elem_dict['picture'] = elem.find_element_by_css_selector(
            'div.picture img').get_attribute('src') ❺
except NoSuchElementException:
    elem_dict['picture'] = None ❻
all_data.append(elem_dict)
```

❶ 这行代码导入来自 Selenium 的 `NoSuchElementException` 异常类。当在 `try...except` 代码块中使用异常类时，确保你导入和使用了库异常，以便正确地处理预期误差。我们知道，不是每一个对象都有照片，并且 Selenium 会在找不到我们想要的 `picture` HTML 元素的时候抛出这个异常，所以我们可以使用这个异常来对有和没有图片的气泡区别处理。

❷ 在 `for` 循环中，遍历了内容气泡。对于这其中的每一个 `elem` 对象，通过更深入地遍历树，可以找到其中包含的元素。

❸ 对于每一个文本对象，这行代码调用了 `HTMLElement` 的 `text` 属性，这会抛弃文本中的标签，只返回元素的文本内容。

❹ `HTMLElement` 的 `get_attribute` 方法期待得到一个嵌套的属性，并且返回属性的值。这行代码使用 `href` 属性来得到 URL，使用嵌套的 CSS 在 `when` 类中的 `div` 元素中查找锚标签。

❺ 在 `try` 代码块中，这段代码在 `div` 中查找照片。如果没有照片，下一行会捕获 Selenium 抛出的 `NoSuchElementException` 异常，因为没有匹配的元素。

❻ 如果没有找到匹配的元素，这行代码添加一个 `None` 值。这确保新列表中的所有对象有一个 `picture` 值。

我们的脚本很早就遇到了问题，你应该会看到一个包含下面文本信息的异常：

```
Message: Unable to locate element:
  {"method":"css selector","selector":"div.when"}
```

这告诉我们在查找 when 元素时碰到了问题。让我们通过检视标签仔细查看发生了什么（见图 12-2）。



图 12-2：相邻 div

通过更仔细地观察，可以看到 content div 和 when div 实际上是相邻的，而不是在 DOM 结构中的父子关系。这暴露了一个问题，因为只遍历了 content div，而不是父 div。如果仔细地观察，可以看到 twine-item-border 是 content 和 when 元素的共同父元素。你需要通过加载父元素，修改为 all_bubbles 使用的元素：

```
all_bubbles = browser.find_elements_by_css_selector('div.twine-item-border')
```

做出这个改变后重新运行之前的代码。发生了什么？你会看到更多的 NoSuchElementException 错误。因为不确定每一个元素有相同的属性，所以我们假设它们都不相同，并且重新编写代码来对异常做出解释：

```
from selenium.common.exceptions import NoSuchElementException

all_data = []
all_bubbles = browser.find_elements_by_css_selector(
    'div.twine-item-border')

for elem in all_bubbles:
    elem_dict = {'full_name': None,
                 'short_name': None,
                 'text_content': None,
                 'picture': None,
                 'timestamp': None,
                 'original_link': None,
                 } ❶
    content = elem.find_element_by_css_selector('div.content') ❷
    try:
        elem_dict['full_name'] = \
            content.find_element_by_css_selector('div.fullname').text
    except NoSuchElementException:
        pass ❸
    try:
        elem_dict['short_name'] = \
```

```
            content.find_element_by_css_selector('div.name').text
    except NoSuchElementException:
        pass
    try:
        elem_dict['text_content'] = \
            content.find_element_by_css_selector('div.twine-description').text
    except NoSuchElementException:
        pass
    try:
        elem_dict['timestamp'] = elem.find_element_by_css_selector(
            'div.when').text
    except NoSuchElementException:
        pass
    try:
        elem_dict['original_link'] = \
            elem.find_element_by_css_selector(
                'div.when a').get_attribute('href')
    except NoSuchElementException:
        pass
    try:
        elem_dict['picture'] = elem.find_element_by_css_selector(
            'div.picture img').get_attribute('src')
    except NoSuchElementException:
        pass
    all_data.append(elem_dict)
```

❶ 对于对象的每一次迭代，这行代码添加一个新的字典，设置所有的键为 `None`。这给了我们一个干净的字典设置，这样每一个对象都有相同的键，我们可以在发现数据的时候添加它到键中。

❷ 拉取 `content div`，这样可以在这个 `div` 中选择。这让代码更加明确，以防有其他的 `div` 有相似的名称。

❸ 使用 Python 的 `pass`（https://docs.python.org/2/tutorial/controlflow.html#pass-statements）来略过异常。因为所有的键都已经设置为 `None`，所以我们在这里不需要做任何事。Python 的 `pass` 让代码略过异常，所以程序会继续执行下一个代码块。

一旦将数据收集到 `all_data` 中，你可以打印它，看一下收集到的内容。下面是一些样例输出（这是一个社交媒体时间线，所以你的时间线会和下面展示的有所不同）：

```
[{'full_name': u'Stefan Brand',
  'original_link': None,
  'picture': u'https://pbs.twimg.com/media/COZlle9WoAE5pVL.jpg:large',
  'short_name': u'',
  'text_content': u'Simply @Fairphone :) #WeAreFairphone http://t.co/vUvKzjX2Bw',
  'timestamp': u'POSTED ABOUT 14 HOURS AGO'},
 {'full_name': None,
  'original_link': None,
  'picture': None,
  'short_name': u'',
  'text_content': None,
  'timestamp': None},
 {'full_name': u'Sietse/MFR/Orphax',
  'original_link': None,
  'picture': None,
```

```
    'short_name': u'',
    'text_content': u'Me with my (temporary) Fairphone 2 test phone.
    # happytester #wearefairphone @ Fairphone instagram.com/p/7X-KXDQzXG/',
    'timestamp': u'POSTED ABOUT 17 HOURS AGO'},...]
```

我们的数据看起来有一些混乱。for 循环很晦涩，很难阅读和理解。同样，看起来我们可以改进一些数据收集方式——我们的日期对象只是一个字符串，但是它更应该是一个日期。我们还需要实验 Selenium 的能力来与页面交互，这可能会帮助我们加载更多的内容。

还需要调试看到的错误。我们找不到正确的短名称；代码看起来返回了一个空字符串。在研究了页面之后，看起来 name div 是隐藏的。在 Selenium 中，隐藏的元素通常是不能阅读到的，所以需要使用该元素的 innerHTML 属性，这会返回标签中的内容。我们也注意到，时间戳数据存储在 title 属性中，并且 URL 事实上存储在 data-href 中，而不是 href 属性中。

> 随着时间推移，编写首次运行就可成功抓取的代码会变得更简单。预期可能出现的问题也变得更简单。通过研究浏览器的开发者工具，并使用 IPython 进行调试，你可以操作变量，测试可能有效的方法。

在找到所有数据的基础上，要确保正确地格式化脚本。我们想要创建函数，更好地抽象数据提取。相对于从初始页面直接解析 URL，应该简化代码，直接加载页面。通过在浏览器中反复试错发现，可以移除 iframe URL 中的长查询字符串（即，?scroll=auto&cols=4&format=embed&eh=...），并且仍然使用来自社交媒体的嵌入内容加载整个页面。让我们看一下整理和简化后的脚本：

```
from selenium.common.exceptions import NoSuchElementException, \
    WebDriverException
from selenium import webdriver


def find_text_element(html_element, element_css): ❶
    try:
        return html_element.find_element_by_css_selector(element_css).text ❷
    except NoSuchElementException:
        pass
    return None


def find_attr_element(html_element, element_css, attr): ❸
    try:
        return html_element.find_element_by_css_selector(
            element_css).get_attribute(attr) ❹
    except NoSuchElementException:
        pass
    return None


def get_browser():
    browser = webdriver.Firefox()
    return browser
```

```
def main():
    browser = get_browser()
    browser.get('http://apps.twinesocial.com/fairphone')

    all_data = []
    browser.implicitly_wait(10) ❺
    try:
        all_bubbles = browser.find_elements_by_css_selector(
            'div.twine-item-border')
    except WebDriverException:
        browser.implicitly_wait(5)
        all_bubbles = browser.find_elements_by_css_selector(
            'div.twine-item-border')
    for elem in all_bubbles:
        elem_dict = {}
        content = elem.find_element_by_css_selector('div.content')
        elem_dict['full_name'] = find_text_element(
            content, 'div.fullname')
        elem_dict['short_name'] = find_attr_element(
            content, 'div.name', 'innerHTML')
        elem_dict['text_content'] = find_text_element(
            content, 'div.twine-description')
        elem_dict['timestamp'] = find_attr_element(
            elem, 'div.when a abbr.timeago', 'title') ❻
        elem_dict['original_link'] = find_attr_element(
            elem, 'div.when a', 'data-href')
        elem_dict['picture'] = find_attr_element(
            content, 'div.picture img', 'src')
        all_data.append(elem_dict)
    browser.quit() ❼
    return all_data ❽

if __name__ == '__main__':
    all_data = main()
    print all_data
```

❶ 创建一个函数，接受 HTML 元素和 CSS 选择器，返回文本元素。在上一个代码实例中，需要一次又一次地重复代码；现在我们想要创建一个函数，这样可以重复使用它，而不需要在脚本中重新编写代码。

❷ 使用抽象函数变量，返回 HTML 元素的文本。如果没有找到匹配，返回 `None`。

❸ 创建一个函数来找到和返回属性，类似于我们的文本元素函数。这需要 HTML 元素、CSS 选择器，以及我们想要从选择器中拉取的属性，并且为这个选择器返回值或者 `None`。

❹ 使用抽象函数变量找到 HTML 元素，并且返回属性。

❺ 使用 Selenium`browser` 类的 `implicitly_wait` 方法，它接受一个希望浏览器在执行下一行代码前隐式等待的秒数为参数。如果不确定页面是否会立即加载完成，这是个很棒的方法。关于隐式和显式等待有很多很棒的 Selenium 文档（http://selenium-python.readthedocs.io/en/latest/waits.html）。

❻ 传递 CSS 选择器，来获取 `when div` 中一个锚标签的 `abbr` 元素的 `title` 属性，以获取时间戳数据。

❼ 抓取数据结束后，使用 `quit` 方法关闭浏览器。

❽ 返回收集的数据。`__name__ == '__main__'` 代码块允许从命令行执行代码时打印数据，或者我们可以导入函数到 IPython 中，并且运行 `main` 函数返回数据。

尝试从命令行中运行脚本，或者将其导入到 IPython，之后运行 `main` 函数。这次数据看起来更加完整了吗？你还会发现添加了另外一个 `try...except` 代码块。我们注意到，有些时候 Selenium 使用的交互会与页面上的 JavaScript 冲突，使 Selenium 抛出一个 `WebDriverException` 异常。允许页面加载更长的时间，再一次尝试后，可以解决这个问题。

如果在浏览器中访问 URL，你可以看到，随着下拉页面能够加载更多的数据。有了 Selenium，我们同样可以做这件事！让我们看一下 Selenium 可以做的其他漂亮的事。可以尝试在 Google 中搜索 Python 网页抓取库，并且使用 Selenium 与搜索结果交互：

```
from selenium import webdriver
from time import sleep


browser = webdriver.Firefox()
browser.get('http://google.com')

inputs = browser.find_elements_by_css_selector('form input') ❶
for i in inputs:
    if i.is_displayed(): ❷
        search_bar = i ❸
        break

search_bar.send_keys('web scraping with python') ❹

search_button = browser.find_element_by_css_selector('form button')
search_button.click() ❺

browser.implicitly_wait(10)
results = browser.find_elements_by_css_selector('div h3 a') ❻

for r in results:
    action = webdriver.ActionChains(browser) ❼
    action.move_to_element(r) ❽
    action.perform() ❾
    sleep(2)

browser.quit()
```

❶ 需要找到一个输入。Google 和其他的站点一样，在页面的很多地方都有输入框，但是通常来说，只有一个大的可见搜索框。这行代码定位所有的输入表单，这样我们有了一个好的起点。

❷ 这行代码遍历每一个输入，看它们是隐藏的还是显示的。如果 `is_displayed` 返回 `True`，那么有了一个可见的元素。反之，这个循环会继续遍历。

❸ 找到一个显示出来的输入时，将它赋值给变量 `search_bar`，终止循环。这会找到第一个可见的输入，这可能是我们想找的那一个。

❹ 这行代码通过使用 `send_keys` 方法发送键和字符串到选定的元素（在这个例子中，它发送键到搜索框）。这类似于在键盘上输入，但是是用 Python！

❺ Selenium 还可以 `click` 页面上可见的元素。这行代码告诉 Selenium 点击搜索表单的提交按钮，来查看搜索结果。

❻ 为了查看所有的搜索结果，这行代码选择 `div` 中有链接的标题元素，这是谷歌搜索结果页面的结构。

❼ 这段代码遍历每一个结果，利用 Selenium 的 ActionChains 定义一系列的操作，并告诉浏览器执行这些操作。

❽ 这行代码使用 ActionChain 的 `move_to_element` 方法，传递给这个方法想要浏览器访问的元素。

❾ 这行代码调用 `perform`，这意味着浏览器会高亮每一个搜索结果。我们使用一个 `sleep` 函数，这告诉 Python 在执行下一行代码前等待特定的秒数（这里是 2），这样，浏览器不会执行得过快，以免你失去很多乐趣。

喔！现在我们可以找到一个站点，填充一个表单，提交它，并且使用 Selenium ActionChains 来遍历结果。正如你看到的，ActionChains 是在浏览器中执行一系列操作的有效方式。你可以在 Selenium 的 Python 附带文档（http://selenium-python.readthedocs.org/）中探索很多很棒的特性，包括显式的等待（http://selenium-python.readthedocs.io/waits.html#explicit-waits，浏览器可以等待，直到一个特定的元素被加载，而不只是整个页面加载完成）、处理警告（http://selenium-python.readthedocs.io/api.html#module-selenium.webdriver.common.alert）和保存截图（http://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver.save_screenshot），这些对于调试来说很有用处。

现在已经看到 Selenium 的一些能力，你能不能重写我们已经为 #WeAreFairphone 站点编写的代码，并且遍历前 100 个记录？［提示：如果你不想使用 ActionChains 遍历每一个元素，你总是可以使用 JavaScript！ Selenium 驱动器的 `execute_script` 方法允许你执行 JavaScript，就像在浏览器控制台中执行 JavaScript 一样。你可以使用 JavaScript 的 `scroll` 方法（https://developer.mozilla.org/en-US/docs/Web/API/Window/scroll）。Selenium 元素对象同样有一个 `location` 属性，它会返回页面上元素的 `x` 和 `y` 坐标值。］

我们已经学习了如何利用 Selenium 操作和使用浏览器来进行网页抓取，但是，还没有结束！让我们看一下如何使用 Selenium 和无头浏览器。

Selenium和无头浏览器

最流行的无头浏览器工具之一是 PhantomJS（http://phantomjs.org/）。如果你是一个熟练的 JavaScript 开发者，可以直接在 PhantomJS 中构建抓取器。然而，如果你想要使用 Python 尝试一下，可以使用 Selenium 和 PhantomJS。PhantomJS 同 GhostDriver（https://github.com/detro/ghostdriver）一起工作，打开 Web 页面并导航。

为什么使用无头浏览器？无头浏览器（http://en.wikipedia.org/wiki/Headless_browser）可以在服务器上运行。相对于普通的浏览器，它们可以更快地运行和解析页面，并且可以在更多的平台上使用。如果最终想要在服务器上运行基于浏览器的网页抓取脚本，你会想要使用无头浏览器。在 10 分钟之内就可安装并运行一个无头浏览器，而大多数其他浏览器需要更长的时间来正确加载和运行（取决于你使用的功能和部署的方式）。

12.1.2 使用Ghost.py进行屏幕读取

Ghost.py（http://jeanphix.me/Ghost.py/）是一个用于屏幕读取的 WebKit 实现，用来直接与 Qt WebKit（http://doc.qt.io/qt-5/qtwebkit-index.html）交互。Qt WebKit 是一个基于 Qt（https://en.wikipedia.org/wiki/Qt_(software)）的 WebKit 实现，而 Qt 是一个用 C++ 实现的跨平台的应用开发框架。

为了开始使用 Ghost.py，你首先需要安装一些很有效的库。如果你能够安装 PySide（https://pypi.python.org/pypi/PySide），那效果会是最好的，这会允许 Python 同 Qt 通信，给 Python 访问更广泛程序和交互的能力。这个过程会花一些时间，所以在开始运行安装之后，尽情地去给自己做一个三明治吧[2]。

```
pip install pyside
pip install ghost.py --pre
```

使用 Ghost.py 搜索 Python 主页（http://python.org），找到新的抓取文档。开始一个新的 Ghost.py 实例非常简单：

```
from ghost import Ghost


ghost = Ghost() ❶
with ghost.start() as session:
    page, extra_resources = session.open('http://python.org') ❷

    print page
    print page.url
    print page.headers
    print page.http_status
    print page.content ❸

    print extra_resources

    for r in extra_resources:
        print r.url ❹
```

❶ 这行代码调用 `Ghost` 类的会话对象，实例化一个 Ghost 对象来同页面交互。

❷ `Ghost` 类的 `open` 方法返回两个对象，所以这行代码在两个独立的变量中捕获这些对象。第一个对象是用来同 HTML 元素交互的页面对象。第二个对象是页面加载的其他资源列表（你在网络标签中看到的列表）。

❸ 页面对象有很多属性，比如头部、内容、链接和页面上的内容。这行代码打印页面的内容。

❹ 这行代码遍历页面的其他资源，并且打印它们，来看是否有用。有时，这些 URL 是 API 调用，可以利用它们简化数据的访问。

Ghost.py 让我们能够洞察页面使用的资源（在第一次使用 `open` 方法打开页面时，通过一个

注 2：如果你在安装 PySide 时碰到了问题，查看与操作系统相关的项目文档。你可以选择安装 PyQt（http://pyqt.sourceforge.net/Docs/PyQt5/installation.html）。你也可以通过 GitHub 上的安装文档（https://github.com/jeanphix/Ghost.py#installation）检查更新。

元组给出）和真实页面上的许多特性。同样可以使用 .content 属性来加载页面的内容，这样如果想要使用其中一个页面解析器解析它，比如 LXML，我们可以做得到，并且仍然使用 Ghost.py 进行交互。

当前，Ghost.py 的大多数能力在于执行页面上的 JavaScript 代码（不是 jQuery），所以你可能需要打开 Mozilla 开发者网络的 JavaScript 指南（https://developer.mozilla.org/en-US/docs/Web/JavaScript）。这会帮助你更加容易地搜索和找到同 Ghost.py 一起使用的 JavaScript。

由于对于在 Python 主页中搜索抓取库感兴趣，让我们看一下是否可以定位输入框：

```
print page.content.contains('input') ❶

result, resources = session.evaluate(
    'document.getElementsByTagName("input");') ❷

print result.keys()
print result.get('length') ❸
print resources
```

❶ 测试页面上是否存在一个 input 标签（大多数的搜索框是简单的输入对象）。这会返回一个布尔值。
❷ 使用一些简单的 JavaScript 来找到页面上所有以“input”为标签名称的元素。
❸ 打印来看响应中的 JavaScript 数组的长度。

根据 JavaScript 结果，在页面上，只有两个输入。为了确定使用哪一个，看一下第一个是否合适。

```
result, resources = session.evaluate(
    'document.getElementsByTagName("input")[0].getAttribute("id");') ❶

print result
```

❶ 索引结果列表，获取 id 属性。JavaScript 直接给出了元素的 CSS 属性，所以这是一个查看选择元素的相关 CSS 的有用方式。

类似于在 Python 中索引结果，也可以在 JavaScript 中索引它们。我们想要第一个输入元素，之后抓取输入的 CSS id。

甚至可以编写一个 JavaScript for 循环（https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for）来遍历 getElementsByTagName 函数返回的列表，通过这种方式来检查属性。如果你希望在浏览器中尝试 JavaScript，可以使用控制台完成这件事（见图 11-12）。

通过 id 的名称（id-search-field），已经定位了搜索字段元素，现在发送一些数据到这个字段：

```
result, resources = ghost.set_field_value("input", "scraping")
```

这段代码使用 set_field_value 方法，它需要一个选择器（这里就是 "input"），并且发送

给它一个字符串（"scraping"）。Ghost.py 同样有一个 fill 方法（http://jeanphix.me/Ghost.py/#form），它会发送一个值的字典，来填充一系列匹配的表单字段。这在有多个字段要填充时很有用处。现在填充了检索词；让我们看一下是否能够提交查询。我们看到这在一个表单中，所以可以直接尝试进行一次表单提交：

```
page, resources = session.fire("form", "submit", expect_loading=True) ❶

print page.url
```

❶ 这行代码调用 Ghost.py 的 fire 方法，这会触发一个 JavaScript 事件。我们想给表单元素发送一个信号，以提交事件，这样它会提交搜索，并导航至下一页。设置 expect_loading 为 True，这样 Ghost.py 知道我们在等待页面加载。

这有效吗？在测试中，当运行这段代码时，收到了超时响应。我们会在本章后文中讨论超时，但是这意味着 Ghost.py 停止等待响应，因为这花费了太长的时间。当你处理抓取器提交数据任务时，找到一个合适的超时时间，对于保证脚本继续工作是必需的。让我们尝试一个不同的提交方式。Ghost.py 可以同页面元素交互并且点击，让我们尝试一下。

```
result, resources = session.click('button[id=submit]') ❶

print result

for r in resources:
    print r.url ❷
```

❶ Ghost.py 的 click 方法使用 JavaScript 选择器点击对象。这行代码点击 id="submit" 的按钮。

❷ 对于大多数通过 Ghost.py 的交互，你会收到一个结果和一个资源列表。这行代码查看代码交互返回的资源。

嗯——点击提交按钮，我们得到了一个看起来像是控制台的 URL。让我们看一下是否可以看到 Qt WebKit 看到的内容。类似于 Selenium 的 save_screenshot 方法，Ghost.py 允许我们查看页面。

> 在使用无头部或不能脱离代码使用的 WebKit 浏览器时，有时页面表现得会与在普通浏览器中有所不同。当使用 Ghost.py 或 PhantomJS 时，你会想要利用屏幕截图来“查看”无头或 kit 浏览器正在使用的页面。

可以使用 Ghost.py 的 show 方法来“查看”该页面：

```
session.show()
```

你会看到打开了一个新窗口，展示出同抓取器所看到的相同的站点。它应该类似于图 12-3。

喔！我们正在页面的中间。尝试向上滚动，从另外的角度看一下。

```
session.evaluate('window.scrollTo(0, 0);')

session.show()
```

现在它应该看起来类似于图 12-4。

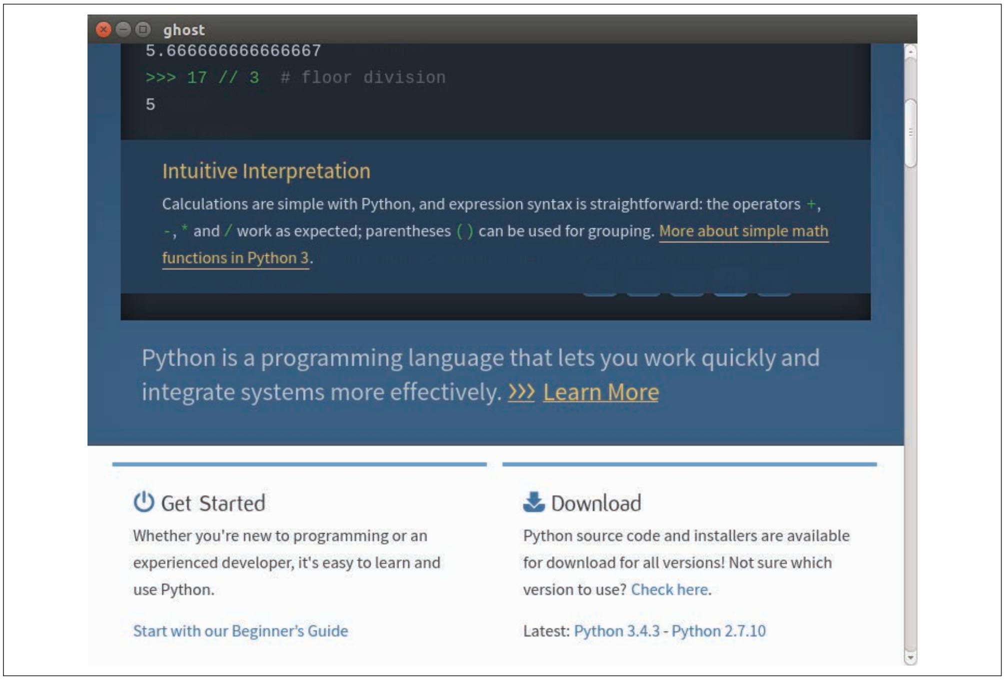


图 12-3：Ghost 页面

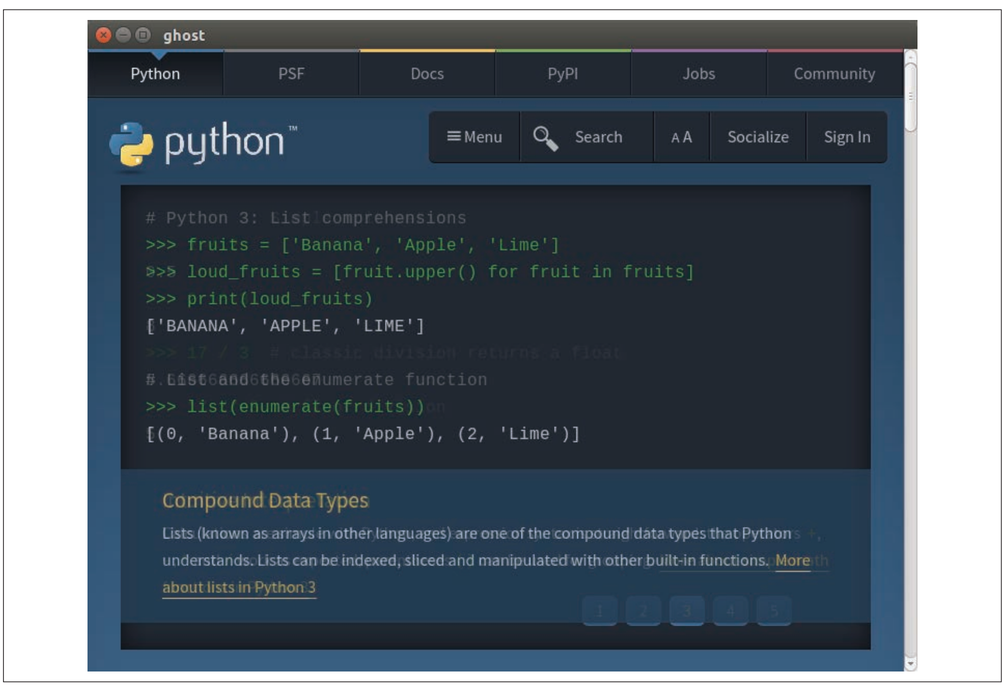


图 12-4：Ghost 页面顶端

这个视图帮助我们理解错误。页面并没有像在普通浏览器中那样完整地打开，搜索和提交输入不可读。一个解决方案是使用更大的视窗重新打开页面，或者为提交设置一个更长的超时时间。

> 正如你可以在文档（http://ghost-py.readthedocs.io/en/latest/index.html）中看到的那样，我们创建的第一个 Ghost 对象可以接受类似 `viewport_size` 和 `wait_timeout` 的参数。如果你希望重启浏览器，设置一个更大的视窗或一个更长的超时时间，这些都是合理的修正。

现在，看看是否可以使用一些 JavaScript 来将它提交：

```
result, resources = session.evaluate(
    'document.getElementsByTagName("input")[0].value = "scraping";') ❶
result, resources = session.evaluate(
    'document.getElementsByTagName("form")[0].submit.click()') ❷
```

❶ 完全使用 JavaScript 设置输入值为“scraping”。
❷ 使用 JavaScript 函数调用表单的提交元素，并主动点击它。

现在如果再一次运行 `show`，你会看到图 12-5 所示的页面。

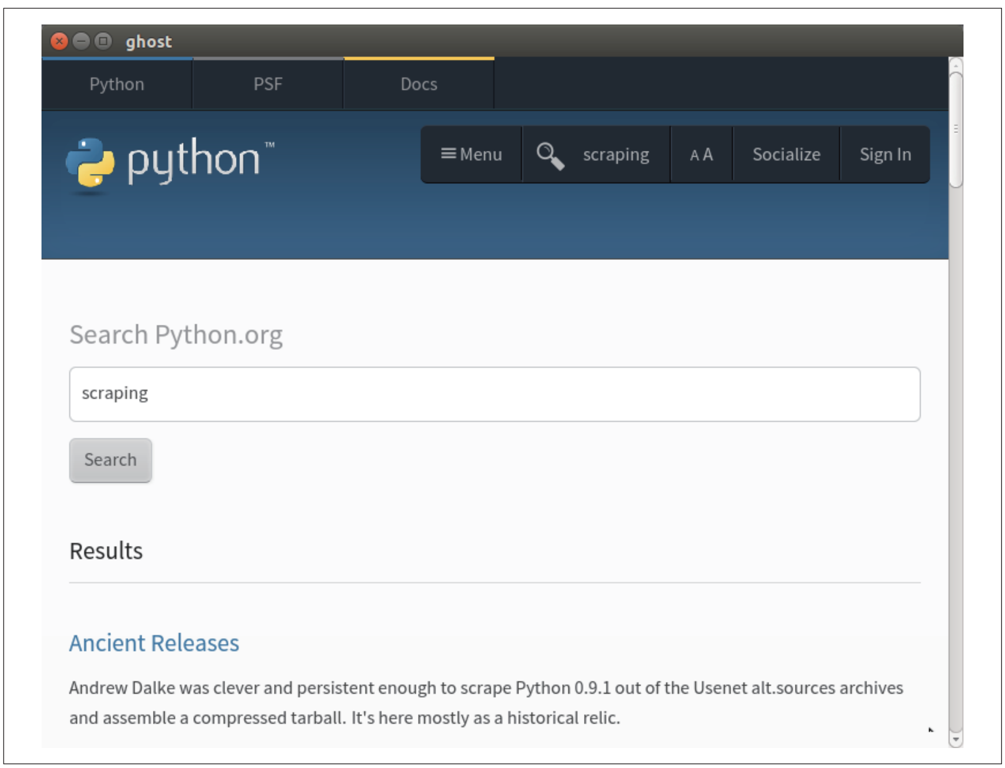


图 12-5：Ghost 搜索

我们使用 Qt 浏览器成功地执行了搜索。一些功能还没有像 Selenium 一样流畅，但是 Ghost.py 仍然是一个相当年轻的项目。

你可以通过评估版本号来评估一个项目的年龄。在编写本书的时候，Ghost.py 仍然低于 1.0 版本（事实上，本书可能只能兼容 0.2 发布版）。它可能会在未来的几年里有大量的改变，但是这是一个非常有趣的项目。我们鼓励你通过向作者提交想法以及研究和修复 bug 来帮助它。

现在，我们已经学习了 Python 中几种与浏览器交互的不同方式，让我们做一些爬取！

12.2 爬取网页

如果你需要从网站的多个页面上抓取数据，爬虫可能是最好的解决方案。网络爬虫（或者机器人）很适合跨越整个域名或站点（或一系列的域名或站点）寻找信息。

你可以将爬虫视为一个高级的抓取器，通过它你可以利用页面读取抓取器的能力（类似于在第 11 章中学到的），并且在整个站点中应用匹配 URL 模式的规则。

爬虫可以帮助你了解网站的结构。例如，站点可能包含一个你并不知道的完整的子章节，其中包含一些有趣的数据。使用爬虫遍历域名，你可以找到子域或其他对报告有用的相关内容。

当你构建爬虫时，首先研究感兴趣的站点，然后创建页面读取的代码来识别和读取内容。一旦爬虫构建完毕，你可以创建一个*遵循的规则列表*，爬虫会使用它找到其他有趣的页面和内容，同时解析器会使用你创建的页面读取抓取器收集和保存内容。

使用爬虫时，你需要事先明确想要什么内容，或者首先使用一个宽泛的方法探索站点，然后重新编写它，使其更明确。如果你选择了广撒网的方法，可能需要在之后做很多数据清洗的工作，以将发现的内容缩小至可用数据集。

我们会使用 Scrapy 开始创建第一个爬虫。

12.2.1 使用Scrapy创建一个爬虫

Scrapy（http://scrapy.org/）是最强大的 Python 网络爬虫。它赋予你在 Python 异步网络引擎 Twisted（http://twistedmatrix.com/trac/）之上使用 LXML（见 11.5 节）的能力。如果你需要一个特别快的爬虫，并能够同时处理大量的任务，我们强烈推荐 Scrapy。

Scrapy 有一些很棒的内置特性，包括导出不同格式的结果（CSV、JSON 等），一个易用的可运行满足不同需求的多个抓取器的服务端部署结构，以及其他一系列优雅的特性，比如使用中间件来处理代理请求或重试状态码失败的请求。Scrapy 会将遇到的错误打印到日志，这样你可以更新和修改代码。

为了恰当地使用 Scrapy，你需要学习 Scrapy 类系统。Scrapy 使用几个不同的 Python 类来

解析网页并且返回好的内容。当定义一个爬虫类时，你也定义了规则和其他的类属性。这些规则和属性在爬虫开始抓取网页时使用。当定义一个新的爬虫的时候，你正在使用一个叫继承（inheritance）的东西。

> **继　　承**
>
> 继承让你能够将一个类作为基类，在其基础上构建额外的属性或方法。
>
> 通过 Scrapy，当继承一个爬虫类时，同样继承了有用的内置方法与属性。之后通过改变一些方法和属性，它们就是专属于你的爬虫了。
>
> Python 的继承是显而易见的：你开始定义一个类，并且将另外的类名称放置在类定义的括号内（例如，`class NewAwesomeRobot(Old Robot):`）。新的类（这里是 `NewAwesomeRobot`）继承自括号内的类（这里是 `OldRobot`）。Python 让我们得以使用这种直接继承，这样当编写新的类时，可以积极地复用代码。
>
> 继承允许我们使用 Scrapy 库中丰富的抓取知识，只需要重新定义一些方法和一些初始化爬虫属性。

Scrapy 使用继承来定义在页面上抓取的内容。对于每一个 Scrapy 项目，你会收集一系列的对象，并且可能会创建一些不同的爬虫。爬虫会抓取页面，使用在设置中定义的任何格式返回对象（即数据）。

相比我们使用过的其他抓取网页的库，使用 Scrapy 爬虫需要更多的组织，但是它相当直观。组织 Scrapy 抓取器便于复用、共享和更新项目。

有一些不同类型的 Scrapy 爬虫，让我们研究一下它们的主要相似点和不同处 。表 12-1 提供了一个总结。

表12-1：爬虫类型

爬虫名称	主要目的	文档
`Spider`	用来解析特定数量的站点和页面	http://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spider.Spider
`Crawl Spider`	遵循一组关于如何解析链接和识别页面内容的正则表达式规则，解析域名	http://doc.scrapy.org/en/latest/topics/spiders.html#crawlspider
`XMLFeed Spider`	用来解析 XML feeds（比如 RSS），从节点中拉取内容	http://doc.scrapy.org/en/latest/topics/spiders.html#xmlfeedspider
`CSVFeed Spider`	用来解析 CSV feeds（或 URL），从行中拉取信息	http://doc.scrapy.org/en/latest/topics/spiders.html#csvfeedspider
`SiteMap Spider`	根据给定的域名列表，解析站点地图	http://doc.scrapy.org/en/latest/topics/spiders.html#sitemapspider

对于通常的网页抓取，你可以使用 `Spider` 类。对于更高级的、遍历整个域名的抓取，使用 `CrawlSpider` 类。如果你有 XML 或 CSV 格式的 feeds 或文件，特别是当它们非常大时，使

用 XMLFeedSpider 和 CSVFeedSpider 来解析它们。如果你需要查看站点地图（你自己的站点或其他站点），使用 SiteMapSpider。

为了进一步熟悉两个主要的类（Spider 和 CrawlSpider），构建一些不同的爬虫。首先，使用一个 Scrapy 爬虫创建一个抓取器来爬取相同的 emoji 表情页面（http://www.emoji-cheat-sheet.com）。为此，我们使用普通的 Spider 类。首先使用 pip 安装 Scrapy。

```
pip install scrapy
```

同样建议你安装 service_identity 模块，这个模块提供了一些好用的特性，在爬取网页时提供安全集成。

```
pip install service_identity
```

通过 Scrapy，你可以使用一个简单的命令来启动一个项目。确保你正在想要使用爬虫的目录下，因为这个命令会为爬虫创建一系列的文件夹和子文件夹：

```
scrapy startproject scrapyspider
```

如果你列出所有当前文件夹下面的文件，应该会看到一个有很多子文件夹和文件的新的父文件夹。正如 Scrapy 站点（https://doc.scrapy.org/en/latest/intro/tutorial.html#creating-a-project）文档中描述的，有一些不同的配置文件（主文件夹下的 scrapy.cfg 和项目文件夹下的 settings.py，以及一个放置爬虫文件的文件夹和一个用来定义对象的文件）。

在创建抓取器之前，需要定义想要在页面数据中收集的对象。打开 items.py 文件（位于项目文件夹的嵌套文件夹内），并且修改它来保存页面数据。

```
# -*- coding: utf-8 -*-
# 在这里定义为要抓取的对象定义模型
#
# 参见文档：
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class EmojiSpiderItem(scrapy.Item): ❶
    emoji_handle = scrapy.Field() ❷
    emoji_image = scrapy.Field()
    section = scrapy.Field()
```

❶ 通过继承 scrapy.Item 创建了新的类。这意味着我们有了这个类的内置方法和属性。

❷ 为了定义每一个字段或数据值，在类中添加了一个新的行，设置了属性名称，并且通过将其设置为 scrapy.Field() 对象来初始化。这些字段支持任何普通的 Python 数据结构，包括字典、元组、列表、浮点数、小数和字符串。

你可能注意到 items.py 文件主要是事先编辑好的。这是个非常好的功能，让你能够快速开始开发，并确保有合适的项目结构。startproject 命令提供所有这些工具，是开始新 Scrapy 项目的最好方式。你同样可以看到，创建一个新的类来收集数据是很简单的。只需要几行 Python 代码，就可以定义关心的域，并准备好爬虫中使用的对象。

为了从爬虫类开始，在新项目目录结构中的 spiders 文件夹下创建一个新的文件，名为 emo_spider.py：

```
import scrapy
from emojispider.items import EmojiSpiderItem ❶


class EmoSpider(scrapy.Spider): ❷
    name = 'emo' ❸
    allowed_domains = ['emoji-cheat-sheet.com'] ❹
    start_urls = [
        'http://www.emoji-cheat-sheet.com/', ❺
    ]

    def parse(self, response): ❻
        self.log('A response from %s just arrived!' % response.url) ❼
```

❶ 所有 Scrapy 导入使用项目根目录作为模块导入起始点，所以需要在导入中包含父文件夹。这行代码从 `emojispider.items module` 导入了 `EmojiSpiderItem` 类。

❷ 使用继承定义了 `EmoSpider` 类，新的类基于简单的 `scrapy.Spider` 类。这意味着爬虫将需要特定的初始化属性（https://doc.scrapy.org/en/latest/topics/spiders.html#spider），这样它知道去抓取哪一个 URL，以及如何处理抓取的内容。我们在下面的几行中定义了这些属性（`start_urls`、`name` 和 `allowed_domains`）。

❸ 爬虫名称是我们在命令行任务中识别出爬虫时会用到的。

❹ `allowed_domains` 告诉爬虫爬取哪些域名。如果爬虫遇到一个链接指向的域名不在该列表中，它会忽略这个链接。这个属性在编写爬取抓取器时很有用，这样如果链接不符合规则，抓取器就不会尝试去爬取所有的 Twitter 或 Facebook 网页。你同时也可以传递子域。

❺ `Spider` 类使用 `start_urls` 属性来遍历要爬取的 URL 列表。在 `CrawlSpider` 里，这些是找到更多匹配的 URL 的起点。

❻ 这行代码重新定义了爬虫的 `parse` 方法，通过在类中使用 `def` 和方法名称定义该方法执行一些逻辑。为类定义方法时，你总是从传递 `self` 开始。这是因为调用方法的对象将是第一个参数（即，`list.append()` 首先传递列表对象本身，然后传递括号中的参数）。`parse` 函数的下一个参数是响应。正如在文档（https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse）中提及的，`parse` 方法将需要一个响应对象。最后用冒号终结这一行，正如定义任何其他的函数时所做的。

❼ 为了测试爬虫，Scrapy 入门指南中的这行代码使用爬虫的 `log` 方法，发送一条信息到日志中。使用响应的 URL 属性来展示响应的地址。

为了运行这个 Scrapy 爬虫，我们要确保正处于恰当的目录中（scrapy spider 和其中的 scrapy.cfg 文件），之后运行命令行参数来解析页面：

```
scrapy crawl emo
```

日志会显示爬虫开始运行，并且显示出哪些中间件正在运行。之后，几乎在最后，你应该能看见类似下面的输出：

```
2015-06-03 15:47:48+0200 [emo] DEBUG: A resp from www.emoji-cheat-sheet.com
    arrived!
```

```
2015-06-03 15:47:48+0200 [emo] INFO: Closing spider (finished)
2015-06-03 15:47:48+0200 [emo] INFO: Dumping Scrapy stats:
    {'downloader/request_bytes': 224,
     'downloader/request_count': 1,
     'downloader/request_method_count/GET': 1,
     'downloader/response_bytes': 143742,
     'downloader/response_count': 1,
     'downloader/response_status_count/200': 1,
     'finish_reason': 'finished',
     'finish_time': datetime.datetime(2015, 6, 3, 13, 47, 48, 274872),
     'log_count/DEBUG': 4,
     'log_count/INFO': 7,
     'response_received_count': 1,
     'scheduler/dequeued': 1,
     'scheduler/dequeued/memory': 1,
     'scheduler/enqueued': 1,
     'scheduler/enqueued/memory': 1,
     'start_time': datetime.datetime(2015, 6, 3, 13, 47, 47, 817479)}
```

抓取器大概一秒解析一个页面。同样可以看到来自 `parse` 方法的日志。酷！我们成功地定义了第一个对象和类，能够创建并且运行它们。

下一步是真正地解析页面，拉取内容。让我们尝试另外一个内置的特性，Scrapy shell。它类似于 Python 或命令行 shell，但是附带所有可用的爬虫命令。有了该 shell，研究页面和确定如何得到页面内容变得非常简单。为了启动 Scrapy shell，只需运行：

```
scrapy shell
```

你应该看到了一个可用选项或可以调用的函数的列表。其中一个为 `fetch`。让我们测试一下这个函数：

```
fetch('http://www.emoji-cheat-sheet.com/')
```

你现在应该看到了一些类似于抓取输出的输出结果。其中一些信息显示已爬取 URL，之后返回一个新的可用对象列表。其中一个是 `response` 对象。响应对象和你在 `parse` 方法中使用的相同。让我们看一下是否可以找到一些同响应对象交互的方式：

```
response.url
response.status
response.headers
```

这其中的每一项都应该返回一些数据。`url` 与我们编写日志信息时使用的 URL 相同。`status` 告诉我们 HTTP 响应的状态码。`headers` 应该提供一个服务器返回的头部字典。

输入 `response`，点击 Tab，你会看到一个完整的可用的方法与属性列表，以及响应对象。你同样可以在 IPython 终端中对任何其他的 Python 对象做这件事。[3]

注 3：如果你安装了 IPython，就会在使用的大多数 Python shell 中看到该 tab 实现。如果没有看到它，你可以添加一个 .pythonrc 文件到计算机（http://stackoverflow.com/questions/246725/how-do-i-add-tab-completion-to-the-python-shell），并且将它赋值给 PYTHONSTARTUP 环境变量。

每一个响应对象同样会有一个 xpath 和 css 方法。这些方法类似于贯穿本章和第 11 章的选择器。正如你已经猜到的，xpath 希望你发送一个 XPath 字符串，而 css 希望得到一个 CSS 选择器。让我们看一下使用已经为这个页面写好的 XPath 选择页面上的一些对象：

```
response.xpath('//h2|//h3')
```

运行该命令，你会看到一个类似于下面的列表：

```
[<Selector xpath='//h2|//h3' data=u'<h2>People</h2>'>,
 <Selector xpath='//h2|//h3' data=u'<h2>Nature</h2>'>,
 <Selector xpath='//h2|//h3' data=u'<h2>Objects</h2>'>,
 <Selector xpath='//h2|//h3' data=u'<h2>Places</h2>'>,
 <Selector xpath='//h2|//h3' data=u'<h2>Symbols</h2>'>,
 <Selector xpath='//h2|//h3' data=u'<h3>Campfire also supports a few sounds<'>]
```

现在让我们看一下，是否可以只读取这些头部中的文本内容。在使用 Scrapy 时，你会想要精确地抽取出正在寻找的元素；（在编写本书时）没有 get 或 text_content 方法。让我们看一下是否可以使用 XPath 知识来从头部中选择文本：

```
for header in response.xpath('//h2|//h3'):
    print header.xpath('text()').extract()
```

你应该会得到类似下面的输出：

```
[u'People']
[u'Nature']
[u'Objects']
[u'Places']
[u'Symbols']
[u'Campfire also supports a few sounds']
```

可以看到 extract 方法会返回一个匹配元素的列表。可以使用 @ 符号来表示属性，用 text() 方法来拉取文本。我们需要重写一些代码，但现在可以使用在 11.5.1 节中所写的许多 LXML 逻辑：

```
import scrapy
from scrapyspider.items import EmojiSpiderItem


class EmoSpider(scrapy.Spider):
    name = 'emo'
    allowed_domains = ['emoji-cheat-sheet.com']
    start_urls = [
        'http://www.emoji-cheat-sheet.com/',
    ]

    def parse(self, response):
        headers = response.xpath('//h2|//h3')
        lists = response.xpath('//ul')
        all_items = [] ❶
        for header, list_cont in zip(headers, lists):
            section = header.xpath('text()').extract()[0] ❷
            for li in list_cont.xpath('li'):
                item = EmojiSpiderItem() ❸
```

```
                item['section'] = section
                spans = li.xpath('div/span')
                if len(spans):
                    link = spans[0].xpath('@data-src').extract() ❹
                    if link:
                        item['emoji_link'] = response.url + link[0] ❺
                    handle_code = spans[1].xpath('text()').extract()
                else:
                    handle_code = li.xpath('div/text()').extract()
                if handle_code:
                    item['emoji_handle'] = handle_code[0] ❻
                all_items.append(item) ❼
        return all_items ❽
```

❶ 由于每个页面使用多个对象，这行代码在 parse 方法的开始使用了一个列表，并在遍历页面的过程中，保存一个找到的对象的列表。

❷ 不同于在 LXML 脚本中调用 header.text，这行代码定位到文本小节（.xpath("text()")），并且使用 extract 函数抽取它。因为我们知道这个方法会返回一个列表，所以这段代码选择每个列表中第一个并且唯一的对象，将其赋值给 section。

❸ 这行代码定义对象。对于每一个列表对象，通过调用类名称与一对空括号创建了一个新的 EmojiSpiderItem 对象。

❹ 为了抽取数据属性，这行代码使用 XPath @ 选择器。这段代码选择第一个 span，并且抽取 @data-src 属性，这会返回一个列表。

❺ 为了创建完整的 emoji_link 属性，这行代码使用响应 URL 并且添加 @data-src 属性中第一个列表对象。为了设置对象的字段，使用字典语法，将键（即字段名称）赋值。如果前面代码没有找到 @data-src，那么这行代码不会执行。

❻ 为了组合一些代码，并且不重复我们自己的代码，这段代码找到 emoji 和声音的处理字符串，赋值给 emoji_handle 字段。

❼ 在列表元素每个循环的最后，这行代码追加新的对象到 all_items 列表。

❽ 在 parse 方法的最后，这行代码返回了所有找到的对象的列表。Scrapy 会在抓取中使用一个返回的对象或对象列表（通常通过保存、清洗或者以我们可以阅读和使用的格式输出数据）。

现在添加了 extract 方法调用，并且更具体地识别出了要从页面中抓取的文本与属性。我们移除了其中的一些 None 逻辑，因为 Scrapy 对象会自动了解拥有哪一个字段，不拥有哪个字段。出于这个原因，如果导出输出到 CSV 或 JSON，它会同时显示空（null）行和找到的值。现在已经更新了同 Scrapy 工作的代码，再一次调用 crawl 方法运行它。

```
scrapy crawl emo
```

你应该看到一些类似于第一个抓取的输出，只是多出了几行！ Scrapy 会在解析网页时打印每一个找到的对象到日志。在最后，你会看到相同的总结输出，显示错误、调试信息和抓取对象的数量。

```
2015-06-03 18:13:51+0200 [emo] DEBUG: Scraped from
    <200 http://www.emoji-cheat-sheet.com/>
    {'emoji_handle': u'/play butts',
     'section': u'Campfire also supports a few sounds'}
```

```
2015-06-03 18:13:51+0200 [emo] INFO: Closing spider (finished)
2015-06-03 18:13:51+0200 [emo] INFO: Dumping Scrapy stats:
    {'downloader/request_bytes': 224,
     'downloader/request_count': 1,
     'downloader/request_method_count/GET': 1,
     'downloader/response_bytes': 143742,
     'downloader/response_count': 1,
     'downloader/response_status_count/200': 1,
     'finish_reason': 'finished',
     'finish_time': datetime.datetime(2015, 6, 3, 16, 13, 51, 803765),
     'item_scraped_count': 924,
     'log_count/DEBUG': 927,
     'log_count/INFO': 7,
     'response_received_count': 1,
     'scheduler/dequeued': 1,
     'scheduler/dequeued/memory': 1,
     'scheduler/enqueued': 1,
     'scheduler/enqueued/memory': 1,
     'start_time': datetime.datetime(2015, 6, 3, 16, 13, 50, 857193)}
2015-06-03 18:13:51+0200 [emo] INFO: Spider closed (finished)
```

Scrapy 在大约 1 秒的时间里解析 900 多个对象——令人惊讶！在查看日志时，我们看到所有的对象均被解析和添加。没有出现任何的错误；如果有的话，会在最后的输出中看到一个错误数量，类似于 `DEBUG` 和 `INFO` 输出行。

我们还没有通过脚本得到一个真正的文件或输出。可以使用一个内置的命令行参数设置一个。使用一些其他的参数选项尝试重新运行爬虫。

```
scrapy crawl emo -o items.csv
```

在抓取的最后，你的项目根目录中应该有一个 item.csv 文件。如果你打开它，应该会看到所有的数据都被导出到了 CSV 格式中。你同样可以导出 .json 和 .xml 文件，所以尽情地通过改变文件名尝试这些选项。

恭喜，你已经搭建了第一个网络爬虫！只需要几个文件和不到 50 行的代码，你就可以在 1 分钟以内解析一整个页面——超过 900 个对象，输出这些发现到一个简单可阅读并且可以轻松分享的格式文件里。正如你看到的那样，Scrapy 是一个非常强大又极其有用的工具。

12.2.2　使用Scrapy爬取整个网站

我们已经探索了使用 Scrapy shell 和 `crawl` 来爬取普通页面，但是如何利用 Scrapy 的能力和速度来爬取整个站点？为了研究 `CrawlSpider` 的能力，需要首先确定要爬取的内容。让我们尝试寻找 PyPI 主页（http://pypi.python.org）中与抓取相关的 Python 包。首先，查看一下页面，找出我们想要的数据。快速搜索“scrape”（https://pypi.python.org/pypi?:action=search&term=scrape&submit=search）显示了一整个列表的搜索结果，其中的每一个页面都有更多的信息，包括文档、一个相关包的链接、一个支持的 Python 版本的列表和最近下载的数量。

可以围绕这些数据构建一个对象模型。一般来说，如果不是与相同的数据关联，我们会为每一个抓取器创建一个新的项目；但是为了方便使用，我们使用和 emoji 抓取器相同的文件夹。从修改 items.py 文件开始：

```
# -*- coding: utf-8 -*-

# 在这里定义为要抓取的对象定义模型
#
# 参见文档：
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class EmojiSpiderItem(scrapy.Item):
    emoji_handle = scrapy.Field()
    emoji_link = scrapy.Field()
    section = scrapy.Field()


class PythonPackageItem(scrapy.Item):
    package_name = scrapy.Field()
    version_number = scrapy.Field()
    package_downloads = scrapy.Field()
    package_page = scrapy.Field()
    package_short_description = scrapy.Field()
    home_page = scrapy.Field()
    python_versions = scrapy.Field()
    last_month_downloads = scrapy.Field()
```

我们在旧的类下面直接定义了新的对象类。在类之间保留几个空行，这样更容易阅读文件和看到类的不同之处。这里，添加了 Python 包页面中我们感兴趣的一些字段，包括过去一个月的下载量、包的主页、支持的 Python 版本以及版本号。

有了对象定义，可以使用 Scrapy shell 来研究 Scrapely 页面上的内容。Scrapely 是 Scrapy 作者的一个项目，使用 Python 来像屏幕一样阅读 HTML。如果还没有安装它，同样建议安装 IPython，这会确保你的输入和输出看起来和本书中的一样，并且提供了一些其他的 shell 工具。在 shell 中（后文指 `scrapy shell`），需要首先使用下面的命令抓取内容。

```
fetch('https://pypi.python.org/pypi/scrapely/0.12.0')
```

可以尝试从页面顶端的 `breadcrumb` 标签中抓取版本号。它们在 ID 为 `breadcrumb` 的 `div` 中，我们可以编写一些 XPath 来找到它。

```
In [2]: response.xpath('//div[@id="breadcrumb"]')
Out[2]: [<Selector xpath='//div[@id="breadcrumb"]'
        data=u'<div id="breadcrumb">\n              <a h'>]
```

IPython 的 `Out` 信息显示我们已经正确地找到了 breadcrumb `div`。通过在浏览器的检视标签中检查元素，我们看到文本位于 `div` 中的一个锚标签中。我们需要使用 XPath 特化，告诉它通过下面这些行代码去查找子锚标签中的文本：

```
In [3]: response.xpath('//div[@id="breadcrumb"]/a/text()')
Out[3]:
[<Selector xpath='//div[@id="breadcrumb"]/a/text()' data=u'Package Index'>,
 <Selector xpath='//div[@id="breadcrumb"]/a/text()' data=u'scrapely'>,
 <Selector xpath='//div[@id="breadcrumb"]/a/text()' data=u'0.12.0'>]
```

现在可以在最后的 `div` 中看到版本号，在抽取的时候处理最后一个 `div`。使用正则表达式做一些测试，确保版本数据是一个数字（见 7.2.6 节），或者使用 Python 的 `is_digit`（见 7.2.3 节）。

现在看一下如何获取页面中略微复杂的部分：最近一个月的下载量。如果在浏览器中检查了这个元素，你会看到它位于一个 span 中的列表项中的无序列表中。你会注意到，其中没有任何一个元素有 CSS ID 或类。你还会注意到 `span` 不包括实际上的单词"month"（为了便于搜索）。让我们看一下是否可以得到一个有用的选择器。

```
In [4]: response.xpath('//li[contains(text(), "month")]')
Out[4]: []
```

喔，使用 XPath 文本搜索寻找元素是不容易的。然而，在 XPath 中，一个好的技巧是如果你轻微地改变查询，解析相似的对象，有些时候会表现得完全不同。尝试运行这个命令：

```
In [5]: response.xpath('//li/text()[contains(., "month")]')
Out[5]: [<Selector xpath='//li/text()[contains(., "month")]'
        data=u' downloads in the last month\n '>]
```

看到没？为什么一个有效，而其他的却没用呢？因为元素是位于 `li` 元素的 `span`，而其他文本位于 `span` 之后，这迷惑了 XPath 模式搜索的层次。页面结构越复杂，编写一个完美的选择器就越难。我们想要在第二个模式中做的有一点不同——我们说"给我位于 `li` 中且其中包含 month 的文本"，而不是"给我一个拥有 month 文本的 `li` 元素"。这里差别很小，但是处理混乱的 HTML 时，通过尝试不同的选择器处理困难的小节是有用处的。

但是我们真正需要的是包含下载数量的 `span`。可以使用 XPath 关系的魔力在链路上浏览并且定位 `span`。尝试下面的代码。

```
In [6]: response.xpath('//li/text()[contains(., "month")]/..')
Out[6]: [<Selector xpath='//li/text()[contains(., "month")]/..' data=u'<li>\n
        <span>668</span> downloads in t'>]
```

通过使用 `..` 操作符，回退到父节点，这样现在同时有了 `span` 后的文本和 `span` 本身。最后一步是选择 `span`，这样不需要担心剥离文本。

```
In [7]: response.xpath('//li/text()[contains(., "month")]/../span/text()')
Out[7]: [<Selector xpath='//li/text()[contains(., "month")]/../span/text()'
        data=u'668'>]
```

棒！现在有了想要找到的数字，并且它应该在所有的页面上工作，因为它基于页面层次编写，并且没有尝试去"猜测"内容可能位于哪里。

> 使用 XPath 技巧在 shell 中调试和定位你想要使用的对象。随着经验的积累，你在第一次尝试编写选择器就会更容易取得成功，所以鼓励你编写更多的抓取器，通过测试更多不同的选择器进行实验。

首先编写一个能够使用 `Spider` 类正确解析 Scrapely 页面的抓取器，之后将它转换为使用 `CrawlSpider` 类的版本。循序渐进地解决一个有两三个因素的问题是个好方法，在完成一部分任务后再完成下一个部分。因为需要使用 `CrawlSpider` 调试两部分代码（抓取规则以

找到匹配的页面和抓取页面本身），所以首先确认其中的一部分有效是比较好的做法。建议从构建一个抓取器（可以在一两个匹配的页面上工作）开始，之后编写抓取规则来测试爬取逻辑。

下面，看一下 Python 包页面的完整的 `Spider`。将它作为一个新的文件包含在 spiders 文件夹中，同 emo_spider.py 文件一起。我们称它为 package_spider.py。

```
import scrapy
from scrapyspider.items import PythonPackageItem


class PackageSpider(scrapy.Spider):
    name = 'package'
    allowed_domains = ['pypi.python.org']
    start_urls = [
         'https://pypi.python.org/pypi/scrapely/0.12.0',
         'https://pypi.python.org/pypi/dc-campaign-finance-scrapers/0.5.1', ❶
    ]

    def parse(self, response):
        item = PythonPackageItem() ❷
        item['package_page'] = response.url
        item['package_name'] = response.xpath(
            '//div[@class="section"]/h1/text()').extract()
        item['package_short_description'] = response.xpath(
            '//meta[@name="description"]/@content').extract() ❸
        item['home_page'] = response.xpath(
            '//li[contains(strong, "Home Page:")]/a/@href').extract() ❹
        item['python_versions'] = []
        versions = response.xpath(
            '//li/a[contains(text(), ":: Python ::")]/text()').extract()
        for v in versions:
            version_number = v.split("::")[-1] ❺
            item['python_versions'].append(version_number.strip()) ❻
        item['last_month_downloads'] = response.xpath(
            '//li/text()[contains(., "month")]/../span/text()').extract()
        item['package_downloads'] = response.xpath(
            '//table/tr/td/span/a[contains(@href,"pypi.python.org")]/@href' ❼
        ).extract()
        return item ❽
```

❶ 这行代码添加一个我们没有研究过的额外的 URL。使用多个 URL 是一种从 `Spider` 转到 `CrawlSpider` 时快速检查代码整洁性和复用性的好方法。

❷ 对于这个抓取器，每个页面只需要一个对象。这行代码在 `parse` 方法的开始创建了这个对象。

❸ 在你解析时，学习一些关于搜索引擎优化（SEO）的知识是获得易读页面描述的一种很好的方式。大多数站点会为 Facebook、Pinterest 和其他共享信息的网站创建简短的描述、关键词、标题和其他元标签。这行代码为数据收集拉取描述。

❹ 包的“Home Page”URL 位于 `li` 中的一个 `strong` 标签中。一旦找到这个元素，这行代码只选择锚元素中的链接。

❺ 版本号链接位于一个使用 `::` 分隔 Python 和版本号的表单对象中。版本号永远出现在最后，这样这行代码使用 `::` 作为分隔符分割字符串，使用最后的元素。

❻ 这行代码追加版本文本（去除额外的空格）到 Python 版本数组。对象的 python_versions 键现在会保存所有的 Python 版本。
❼ 可以看到，在表格中有使用 pypi.python.org 域名的链接，而不是它们的 MD5 校验值。这行代码判断链接是否有正确的域名，并只抓取有正确域名的链接。
❽ 在 parse 方法的最后，Scrapy 希望我们返回一个对象（或一个对象列表）。这行代码返回这些对象。

运行这段代码（scrapy crawl package），你应该会得到两个对象，并且没有错误。然而，你会发现我们得到一些不同的数据。举个例子，对于每一个下载，我们的包数据没有一个好的支持的 Python 版本的列表。如果想要这个列表，可以从表格中的 PyVersion 字段解析，并将它与每一个下载匹配。你会怎样做这件事呢？（提示：这个字段位于每个数据行的第三列，XPath 允许你传递元素索引。）我们同样注意到数据有一些混乱，就像下面的输出（为了匹配页面进行了格式化；你的输出会看起来有一些不同）所展示的一样。

```
2015-09-10 08:19:34+0200 [package_test] DEBUG: Scraped from
    <200 https://pypi.python.org/pypi/scrapely/0.12.0>
    {'home_page': [u'http://github.com/scrapy/scrapely'],
     'last_month_downloads': [u'668'],
     'package_downloads':
    [u'https://pypi.python.org/packages/2.7/s/' + \
      'scrapely/scrapely-0.12.0-py2-none-any.whl',
     u'https://pypi.python.org/packages/source/s/' + \
      'scrapely/scrapely-0.12.0.tar.gz'],
    'package_name': [u'scrapely 0.12.0'],
    'package_page': 'https://pypi.python.org/pypi/scrapely/0.12.0',
    'package_short_description':
    [u'A pure-python HTML screen-scraping library'],
    'python_versions': [u'2.6', u'2.7']}
```

有几个字段原本希望是字符串或整数值，但是取而代之的是一个字符串数组。让我们在定义爬虫规则之前创建一个辅助方法来清洗数据。

```
import scrapy
from scrapyspider.items import PythonPackageItem


class PackageSpider(scrapy.Spider):
    name = 'package'
    allowed_domains = ['pypi.python.org']
    start_urls = [
        'https://pypi.python.org/pypi/scrapely/0.12.0',
        'https://pypi.python.org/pypi/dc-campaign-finance-scrapers/0.5.1',
    ]

    def grab_data(self, response, xpath_sel): ❶
        data = response.xpath(xpath_sel).extract() ❷
        if len(data) > 1: ❸
            return data
        elif len(data) == 1:
            if data[0].isdigit():
                return int(data[0]) ❹
            return data[0] ❺
```

```
        return [] ❻

    def parse(self, response):
        item = PythonPackageItem()
        item['package_page'] = response.url
        item['package_name'] = self.grab_data(
            response, '//div[@class="section"]/h1/text()') ❼
        item['package_short_description'] = self.grab_data(
            response, '//meta[@name="description"]/@content')
        item['home_page'] = self.grab_data(
            response, '//li[contains(strong, "Home Page:")]/a/@href')
        item['python_versions'] = []
        versions = self.grab_data(
            response, '//li/a[contains(text(), ":: Python ::")]/text()')
        for v in versions:
            item['python_versions'].append(v.split("::")[-1].strip())
        item['last_month_downloads'] = self.grab_data(
            response, '//li/text()[contains(., "month")]/../span/text()')
        item['package_downloads'] = self.grab_data(
            response,
            '//table/tr/td/span/a[contains(@href,"pypi.python.org")]/@href')
        return item
```

❶ 这行定义了一个新的方法以使用 `self` 对象（这样爬虫可以像普通方法一样调用它）、响应对象以及长长的 XPath 选择器来查找内容。

❷ 这行代码使用新的函数变量抽取数据。

❸ 如果数据的长度大于 1，这行代码返回列表。我们可能想要所有的数据，所以原样返回。

❹ 如果数据的长度等于 1，并且数据是一个数字，这行代码返回整数。这可能会是下载数量的情况。

❺ 如果数据的长度等于 1，但是不是一个数字，这行代码只返回这个数据。这会匹配包含链接和简单文本的字符串。

❻ 如果函数没有返回，这行代码返回一个空列表。这里使用一个列表，因为我们希望 `extract` 在没有找到数据的时候返回空列表。如果使用 `None` 类型或者空字符串，你可能需要修改其他的代码，来保存它到 CSV。

❼ 这行代码调用新的函数，并且使用下面的参数触发 `self.grab_data`：响应对象和 XPath 选择字符串。r 使用其他内置输出功能。

现在我们有了相当干净的数据和代码，并且更少地重复自己的代码。我们可以更加深入地优化它，但是为了不让你眼花缭乱，先来定义爬取规则。爬取规则由正则表达式实现，通过定义页面位置和遵循的 URL 类型，告诉爬虫去哪里爬取。（第 7 章介绍了正则表达式，是不是很棒？你现在已经是专业的了！）如果看一下包的链接（https://pypi.python.org/pypi/dc-campaign-finance-scrapers/0.5.1 和 https://pypi.python.org/pypi/scrapely/0.12.0），可以看到以下相似点。

- 它们都有相同的域名，pypi.python.org，并且它们都使用 https。
- 在 URL 中，它们都有相同的路径模式：/pypi/<name_of_the_library>/<version_number>。
- 库的名称使用小写字母和破折号，版本号由数字和句点组成。

可以使用这些相似性来定义正则规则。在脚本中编写它们之前，先在 Python 控制台中尝试它们。

```
import re

urls = [
    'https://pypi.python.org/pypi/scrapely/0.12.0',
    'https://pypi.python.org/pypi/dc-campaign-finance-scrapers/0.5.1',
]

to_match = 'https://pypi.python.org/pypi/[\w-]+/[\d\.]+' ❶

for u in urls:
    if re.match(to_match, u):
        print re.match(to_match, u).group() ❷
```

❶ 这行代码找到一个使用 https、域名为 pypi.python.org 并且还有我们研究路径的链接。第一个部分是 pypi，第二个是有着符号“-”的小写文本（使用 [\w-]+ 可以轻松地匹配），最后一部分寻找有或没有句点的数字（[\d\.]+）。

❷ 这行代码输出了匹配的组。我们正在使用正则表达式的 match 方法，因为这是正则 Scrapy 爬虫所使用的。

我们有了一个匹配（确切地说，是两个！）。现在，最后看一下需要从哪里开始。Scrapy 爬虫会首先使用起始 URL 列表，然后跟随这些网页找到其他 URL。如果再看一下搜索结果页（https://pypi.python.org/pypi?:action=search&term=scrape&submit=search），我们会注意到页面使用相对 URL，这样只需要匹配 URL 路径。我们同样看到所有的链接都位于表格中，这样可以限制 Scrapy 查看以找到用来爬取链接的位置。知道这些后，通过添加爬取规则来更新文件。

```
from scrapy.contrib.spiders import CrawlSpider, Rule ❶
from scrapy.contrib.linkextractors import LinkExtractor ❷
from scrapyspider.items import PythonPackageItem

class PackageSpider(CrawlSpider): ❸
    name = 'package'
    allowed_domains = ['pypi.python.org']
    start_urls = [
        'https://pypi.python.org/pypi?%3A' + \
                'action=search&term=scrape&submit=search',
        'https://pypi.python.org/pypi?%3A' + \
                'action=search&term=scraping&submit=search', ❹
    ]
    rules = (
        Rule(LinkExtractor(
            allow=['/pypi/[\w-]+/[\d\.]+', ], ❺
            restrict_xpaths=['//table/tr/td', ], ❻
        ),
            follow=True, ❼
            callback='parse_package', ❽
        ),
    )
```

```
    def grab_data(self, response, xpath_sel):
        data = response.xpath(xpath_sel).extract()
        if len(data) > 1:
            return data
        elif len(data) == 1:
            if data[0].isdigit():
                return int(data[0])
            return data[0]
        return []

    def parse_package(self, response):
        item = PythonPackageItem()
        item['package_page'] = response.url
        item['package_name'] = self.grab_data(
            response, '//div[@class="section"]/h1/text()')
        item['package_short_description'] = self.grab_data(
            response, '//meta[@name="description"]/@content')
        item['home_page'] = self.grab_data(
            response, '//li[contains(strong, "Home Page:")]/a/@href')
        item['python_versions'] = []
        versions = self.grab_data(
            response, '//li/a[contains(text(), ":: Python ::")]/text()')
        for v in versions:
            version = v.split("::")[-1]
            item['python_versions'].append(version.strip())
        item['last_month_downloads'] = self.grab_data(
            response, '//li/text()[contains(., "month")]/../span/text()')
        item['package_downloads'] = self.grab_data(
            response,
            '//table/tr/td/span/a[contains(@href,"pypi.python.org")]/@href')
        return item
```

❶ 这行代码同时导入 `CrawlSpider` 类和 `Rule` 类，因为在第一个爬虫中，我们需要它们。

❷ 这行代码导入了 `LinkExtractor`。默认的链接抽取器使用 LXML（我们知道如何编写它！）。

❸ 这行代码重新定义了 `Spider`，这样它从 `CrawlSpider` 类继承而来。由于修改了这种继承，需要定义一个 `rules` 属性。

❹ 包含了检索词为 scrape 和 scraping 的搜索页，以查看是否可以找到更多的 Python 包。如果你有不同的让脚本开始搜索的起始点，可以在这里添加一个长列表。

❺ 这行代码设置了 `allow` 来使用正则匹配页面上的链接。因为只需要相关的链接，所以只从匹配的链接开始。`allow` 接受一个列表，所以如果你有不止一个类型的 URL 想要匹配，可以在这里添加多个 `allow` 规则。

❻ 这行代码限制了爬虫到结果表格中。这意味着爬虫只会去表格列中的数据行寻找匹配链接。

❼ 这行告诉了跟随（即加载）匹配链接的规则。有些时候，对于一些页面你可能只想解析并获取内容，但是不需要跟随它的链接。如果想要让爬虫跟随页面链接，并且打开它们，你需要使用 `follow=True`。

❽ 赋值给规则一个回调函数，并且重新命名 `parse` 方法来确认没有与 Scrapy 的 `CrawlSpider` 类使用的解析方法混淆 。现在解析方法叫作 `parse_package`，并且爬虫在跟随匹配的 URL 拿到我们想要抓取的页面后会调用这个方法。

你可以同运行一个普通的抓取器一样，运行这个爬虫：

```
scrapy crawl package
```

你已经正式地完成了第一个爬虫！是否还有待完善的地方？有一个容易修复的 bug 遗留在这段代码中了。你能找到它吗？如何修复它？［提示：查看你的 Python 版本，然后查看返回版本的方式（即永远返回一个列表），与 `grab_data` 返回数据对比。］看看你是否能够在爬虫脚本中修复这个问题。如果不能，可以参考本书仓库（https://github.com/jackiekazil/data-wrangling），得到完整的修复后的代码。

Scrapy 是一个有效、快速、方便配置的工具。还有很多值得探索，你可以阅读该库的很棒的文档（http://doc.scrapy.org/en/latest/）。配置你的脚本来使用数据库和特殊的信息抽取工具，并且在自己的服务器上使用 Scrapyd（http://scrapyd.readthedocs.org/en/latest/）运行它们是很简单的。希望这是你之后众多 Scrapy 项目的第一个！

现在你理解了屏幕读取器、浏览器读取器和爬虫。让我们看看构建更加复杂的网页爬虫所需要知道的其他一些事情。

12.3 网络：互联网的工作原理，以及为什么它会让脚本崩溃

取决于运行抓取脚本的频率，以及每个脚本工作的重要性，你可能会碰到网络问题。是的，互联网正在尝试破坏你的脚本。为什么？因为互联网认为如果你真的在乎，你会重试。在网页抓取世界里丢失的连接、代理问题以及超时问题普遍存在。然而，有一些方法可以缓解这些问题。

在浏览器中，如果有页面信息没有正确加载，你就会点击刷新，立即发送另一个请求。对于抓取器，你可以模仿这种行为。如果你正在使用 Selenium，刷新内容会极其简单。Selenium 的 `webdriver` 对象有一个 `refresh` 函数，就像浏览器一样。如果你已经填充了一个表单，需要重新提交表单，前进至下一页（有时这类似于浏览器的行为）。如果你需要同警告或弹出窗口交互，Selenium 提供了接受或拒绝信息所需的工具。

Scrapy 有内置的重试中间件。要启用它，你只需将它添加到项目的 settings.py 文件的中间件列表中。中间件（https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#module-scrapy.contrib.downloadermiddleware.retry）希望你在设置中设定一些默认值，以便它知道哪些 HTTP 响应码需要重试（例如，它需要只在返回码为 500 的时候重试吗？），以及重试的次数。

如果你没有指定这些值，它仍然会使用文档中列出的默认值工作。如果你看到网络错误，然后下载等待时间（另外一个全局设置变量）变长，建议你从 10 次重试开始，或者检查收到的返回码，看看脚本是否访问网站过于频繁。

如果你正在使用自己的 Python 脚本和 LXML 或 BeautifulSoup，最好是捕获这些错误并确定处理它们的方法。大多数时间里，你会注意到 `urllib2.HTTPError`（https://docs.python.org/2/library/urllib2.html#urllib2.HTTPError）异常的优势；或者，如果你正在使用

requests，代码不会加载内容，并且失败。在 Python 中使用一个 try...except 代码块，你的代码可能看起来像下面一样。

```
import requests
import urllib2


resp = requests.get('http://sisinmaru.blog17.fc2.com/')

if resp.status_code == 404: ❶
    print 'Oh no!!! We cannot find Maru!!'
elif resp.status_code == 500:
    print 'Oh no!!! It seems Maru might be overloaded.'
elif resp.status_code in [403, 401]:
    print 'Oh no!! You cannot have any Maru!'


try:
    resp = urllib2.urlopen('http://sisinmaru.blog17.fc2.com/') ❷
except urllib2.URLError: ❸
    print 'Oh no!!! We cannot find Maru!!'
except urllib2.HTTPError, err: ❹
    if err.code == 500: ❺
        print 'Oh no!!! It seems Maru might be overloaded.'
    elif err.code in [403, 401]:
        print 'Oh no!! You cannot have any Maru!'
    else:
        print 'No Maru for you! %s' % err.code ❻
except Exception as e: ❼
    print e
```

❶ 当使用 requests 库来查找网络错误时，检查响应的 status_code。这个属性会返回一个代表在 HTTP 响应中收到的代码的整数。这行代码测试响应是否为 404 错误。

❷ 如果正在使用 urllib2，将请求放在一个 try 语句中（正如这行代码一样）。

❸ 可能会从 urllib2 中看到的一个异常是 URLError。编写一个捕获方法是好的想法。如果它不能解析域名，可能会抛出这个错误。

❹ 可能看到的另外一个异常是 HTTPError。任何有关 HTTP 请求错误的响应都会抛出这个错误。通过添加冒号和 err，捕获了错误，并且将它保存在变量 err 中，这样可以打印错误到日志。

❺ 现在捕捉到了错误，并且将其赋值给前行代码中的 err，这行代码判断 code 属性，来查看 HTTP 错误码。

❻ 对于所有其他的 HTTP 错误，这行代码使用 else 通过格式化它到字符串，来展示错误码。

❼ 这行代码捕获其他所有可能碰到的错误，并且展示错误信息。赋值异常给变量 e，并且打印它，这样可以阅读异常信息。

巧妙地设计脚本，让它尽可能地抗失败，这是一个重要的步骤（第 14 章会更详细地讨论）；同时确保在代码中有合适的 try...except 代码块来解释错误，也是进程中重要的一部分。除了 HTTP 错误，有些时候，页面花费过长的时间来加载。如果抓取器响应缓慢或碰到了延迟问题，我们可能会调整超时时间。

> 什么是延迟？从网络角度来讲，这是数据从一个地点发送到另一个地点需要的时间。往返延迟是从计算机发送请求到服务器，并且得到响应花费的时间。延迟因为数据的传输而存在，有时数据需要传送几千千米，来完成请求。

当编写和规模化脚本时，考虑延迟是很好的。如果脚本所连接的站点托管在另一个国家，你会经历网络延迟。因此你需要相应地调整超时时间，或者创建一个距离目标端点近的服务器。如果想要添加超时时间到 Selenium 和 Ghost.py 脚本，你可以在抓取工作开始的时候，直接添加到脚本中。对于 Selenium，使用 `set_page_load_timeout`（http://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver.set_page_load_timeout）方法，或使用隐式 / 显式的等待（http://selenium-python.readthedocs.io/waits.html），这样浏览器会等待代码中特定的部分加载。对于 Ghost.py，你可能需要传递 `wait_timeout` 参数，像 `Ghost` 类文档中定义的那样（http://ghost-py.readthedocs.io/en/latest/#ghost.Ghost）。

对于 Scrapy，抓取器的异步特性和对特定的 URL 重试若干次的能力，让超时设置变成了一个略微难办的问题。当然，你可以在 Scrapy 设置中使用 `DOWNLOAD_TIMEOUT`（http://doc.scrapy.org/en/latest/topics/settings.html#download-timeout）直接改变超时时间。

如果你正在编写自己的 Python 脚本，并且使用 LXML 或 BeautifulSoup 来解析页面，添加超时到调用是你的职责。如果使用 `requests` 或 `urllib2`，你可以在调用页面的时候直接这样做。在 `requests` 中，你可以直接将其作为一个参数，添加到 `get` 请求中（http://docs.python-requests.org/en/latest/user/quickstart/#timeouts）。对于 `urllib2`，你需要传递超时时间，作为 `urlopen` 方法（https://docs.python.org/2/library/urllib2.html#urllib2.urlopen）的参数之一。

如果你正经历持续的网络方面的问题，并且脚本需要依据一个稳定的日程运行，建议你创建一些日志，尝试在另外一个网络上运行（即不是你的家庭网络，来判断你的家庭互联网连接是否存在问题），并且测试是否在一个非高峰时间段运行会有帮助。

> 脚本在每天下午 5 点还是上午 5 点钟更新对你是否很重要？很可能在下午 5 点钟你的本地互联网服务提供商会非常繁忙，而上午 5 点钟可能会很安静。如果在高峰时段你的家庭网络很难做成事，那么很可能你的脚本也同样做不了什么！

除了网络问题之外，你可能会找到其他破坏抓取脚本的问题，比如互联网在不停变化这一事实。

12.4　变化的互联网（或脚本为什么崩溃）

正如你知道的，网页的重新设计、更新的内容管理系统和页面结构的改变（一个新的广告系统、一个新的推荐网络等）是互联网中很正常的一部分。互联网成长并且变化着。因此，你的网页抓取脚本会崩溃。好消息是，有很多的站点只是每年更新一次，或几年改变一次。还有些改变不会影响页面结构（有时样式更新或广告更新不会改变代码的内容和结

构）。不要彻底失去希望；很可能你的脚本会工作很长一段时间！

无论怎样，我们不想给你虚假的希望。你的脚本最终会崩溃。总有一天，你会继续运行它，然后发现它不再工作了。当发生这些情况时，给自己一个大大的拥抱，为自己冲一杯茶或咖啡，然后重新开始。

现在你知道了更多关于检验网站上的内容和为报告找出最有用的那部分的方法。你已经有了相当多的代码，大部分仍然能够工作。你现在处在一个好的调试阶段，并且有很多工具任你使用，以找到新的 `div` 或包含所需数据的表。

12.5 几句忠告

当抓取网页时，谨慎是很重要的。你还需要了解所在国家关于网页内容的法律。一般来说，如何做到谨慎是很显然的。不要把别人的内容当作自己的来用。不要使用已经声明不允许分享的内容。不要向别人或网站发送垃圾邮件。不要攻击网站或恶意地爬取站点。最基本地，不要做一个蠢人！如果你不能同母亲或其他亲近的人分享正在做的事情，并且感觉良好，那就不要做。

有几种方式来明确你在互联网上做的事情。许多抓取库允许你发送 `User-Agent` 字符串。你可以将自己的信息或者公司的信息放到这些字符串中，这样抓取者的信息就很清晰。同时，确保查看站点的 robot.txt 文件（http://www.robotstxt.org/robotstxt.html），它会告诉网页抓取器站点中禁止爬取的内容。

> 在构建爬虫遍历一个站点之前，看一下站点中你感兴趣的部分是否包含在 robot.txt 的 `Disallow` 小节中。如果它们存在其中，你需要找到别的方式来获得数据，或者联系站点的拥有者，看看他们是否会通过其他方式为你提供数据。

在互联网上规规矩矩做好，并且在构建抓取器时做正确的事。这意味着你可以为自己的工作感到骄傲；不要麻烦律师、公司和政府；放心地使用收集的信息。

12.6 小结

现在在为难于解析内容的网页编写抓取器时，你应该感到胸有成竹。你可以使用 Selenium 或 Ghost.py 打开浏览器，读取一个网页，同页面交互，并且抽取数据。你可以使用 Scrapy 来爬取整个域名（或一系列域名），并且抽取大量的数据。你同样可以练习正则表达式语法，编写自己的 Python 类（在 Scrapy 的帮助下）。

掌握这些后，Python 代码就会顺其自然地完成了。你探索了一些 bash 命令。你积累了一些非常棒的与 shell 脚本交互的经验，正在成为一名专业的数据处理者。表 12-2 列出了本章中引入的新概念和工具。

表12-2：新的Python编程概念与库

概念/库	目的
Selenium 库	该库用于直接同网页和它们的元素交互，你可以使用所选的浏览器，也可以使用无头浏览器。在你需要点击元素、在表单中输入信息，并且同需要几个请求来加载内容的页面交互时，表现良好
PhantomJS 库	JavaScript 库，作为无头浏览器，用于在服务器或无浏览器机器上进行网页抓取。还可用来只使用 JavaScript 编写抓取器
Ghost.py 库	通过 Qt WebKit 而不是通过传统的浏览器来与网页交互的库。可以在需要浏览器等相似条件下使用，有编写原生 JavaScript 的能力
Scrapy 库	用于跨越一个域名或多个不同域名爬取大量网页。在你需要研究多个域名或多种页面类型来收集数据时很有用处
Scrapy 爬取规则	爬取规则告诉你的爬虫匹配 URL 结构，并识别出可能存在该 URL 的页面位置。这使得爬虫可以浏览并找到更多信息

最后，对于抓取器，确保你遵循一些基本的逻辑（见表 12-3）。

表12-3：使用哪一个抓取器

抓取器类型	库	使用场景
页面读取抓取器	BeautifulSoup、LXML	简单页面抓取，你想要的所有数据在一次请求后全部加载于页面上
基于浏览器的抓取器	Selenium、PhantomJS、Ghost.py	基于浏览器的抓取，你需要同页面上的元素交互，或这个页面需要不同的请求加载
Web 爬虫	Scrapy	用快速和异步的方式跨越多个页面跟随链接或解析相似的页面。如果你需要跨越整个域名或一系列域名的多个匹配，这会很有用

后面几章关注如何拓展使用 API 的 Web 技巧，以及规模化和自动化数据。这些是将学到的所有知识整合为一系列可复用、可执行的脚本的最后几步——其中一些脚本不需要做任何事情就可运行。还记得你在开始阅读本书时想到的那些死记硬背的任务吗？以后再也不必死记硬背它们了——继续阅读！

第 13 章

应用编程接口

应用编程接口（application programming interface，API）听起来是一个很酷炫的概念，但事实上并不是。API 是在 Web 上共享数据的一种标准方式。许多网站通过 API 端点来共享数据。本书列出了很多可用的 API，但是下面是一些很有用或有趣的 API。

- Twitter（https://dev.twitter.com/overview/api）
- US Census（http://www.census.gov/data/developers/data-sets.html）
- World Bank（http://data.worldbank.org/node/9）
- LinkedIn（https://developer.linkedin.com/docs/rest-api）
- San Francisco Open Data（https://data.sfgov.org/）

这些都是返回数据的 API 示例。你向 API 发出请求，随后 API 返回数据。API 同样可以作为和其他应用交互的一种方式。例如，我们可以使用 Twitter API 来获得 Twitter 的数据，构建另外一个和 Twitter 交互的应用（例如，一个使用 API 发布推文的应用）。Google API 列表（https://developers.google.com/apis-explorer/#p/）是另外一个示例，大多数 API 允许你同公司服务进行交互。使用 LinkedIn API，你可以获取数据，发布更新到 LinkedIn，而不需要通过访问 Web 界面。因为 API 可以做很多不同的事情，所以它应该被认为是一种服务。就我们的目的而言，这个服务提供数据。

在这一章中，你会请求 API 数据并且保存到电脑。API 通常会返回 JSON、XML 或 CSV 文件，这意味着在数据保存到本地计算机之后，你需要应用在本书前几章学到的知识解析这些数据。本章使用的 API 是 Twitter API。

我们出于下面几个原因选择 Twitter API 作为样例。首先，Twiiter 是一个众所周知的平台。其次，Twitter 拥有大量的数据（推文），人们对分析这些数据有很大的兴趣。最后，Twitter API 允许我们探索许多 API 的概念，这正是本章讨论的话题。

Twitter 数据不仅是非正式的信息收集工具，类似于 One Million Tweet Map（http://onemilliontweetmap.com/），同时也是更正式的学术研究工具，例如用于预测流感趋势（http://ieeexplore.ieee.org/document/5928903/?reload=true&tp=&arnumber=5928903&url=http:%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5928903），以及捕获实时事件的发生，例如地震（http://dl.acm.org/citation.cfm?id=1772777）。

13.1 API特性

API 可以像数据响应请求一样简单，但是很难找到只有这个功能的 API 了。大多数的 API 还有其他有用的特性。这些特性包括多种不同的 API 请求方法（REST 或流式请求）、数据时间戳、频率限制、数据分级，以及 API 访问对象（key 和 token）。让我们通过 Twitter API 的上下文看一下这些特性。

13.1.1 REST API与流式API

Twitter API 有两种形式：REST API 和流式 API。大多数的 API 是 REST 形式的，但是一些实时服务通常会提供流式 API。REST 的全称为 Representational State Transfer（*表述性状态转移*），被设计用来构建稳定的 API 架构。可以使用 `requests` 库（见第 11 章）来获取 REST API 的数据。通过 `requests` 库，你可以发出 `GET` 和 `POST` Web 请求——这是 REST API 用来返回对应数据所使用的方法。在 Twitter 这个实例中，REST API 允许你查询推文，发布推文，做 Twitter 允许通过网站做的绝大多数事情。

通过 REST API，你经常可以（但并不总是可以）在浏览器中通过请求 API 链接预览查询。如果在浏览器中加载了 URL，结果看起来像是文本块，你可以在浏览器中安装一个格式化预览插件。例如，Chrome 有 JSON 文件的预览插件，通过一种更好阅读的方式预览 JSON 内容。

流式 API 作为一种实时服务运行，监听对相关数据的请求。碰到流式 API 时，你会想要使用一个构建好的库帮助管理数据的拉取。想详细了解多 Twitter 流式 API 是如何工作的，可以查看 Twitter 网站上的概述（https://dev.twitter.com/streaming/overview）。

13.1.2 频率限制

API 通常都有频率限制，限制用户在一段时间内能够请求的数据数量。频率限制由 API 提供者出于多种不同的原因使用。除了频率控制，你可能还会碰到对数据访问有限制的 API，特别是数据与商业利益相关时。出于基础设施和客户服务的考虑，API 提供者会想要控制请求的数量，这样服务器和架构可以控制数据传输数量。如果允许每个人在 100% 的时间使用 100% 的数据，这可能会导致 API 服务的崩溃。

如果遇到需要支付额外费用来取得特殊访问权限的 API，你需要确定是否有能力支付，以及数据对研究有多大的价值。如果碰到有频率限制的 API，你需要确定是否数据的一个子集就足够了。如果 API 有频率限制，可能会花费相当长的时间来收集一个有代表性的样

例，所以一定要评估你想要投入在这之上的努力程度。

API 通常会对所有的用户有频率控制，因为这样更易于管理。Twitter 的 API 曾经也使用这种方式；然而，随着流式 API 的出现，用法发生了改变。Twitter 的流式 API 提供了数据的常量流，而 REST API 限制了每 15 分钟你可以发出请求的数量。为了帮助开发者理解频率限制，Twitter 发布了一张图表（https://dev.twitter.com/rest/public/rate-limits）。

在练习中，我们会使用名叫获取搜索 / 推文（GET search/tweets）的 API。这个接口返回包含特定搜索语素的推文。如果你查看文档（https://dev.twitter.com/rest/reference/get/search/tweets），会发现 API 返回 JSON 格式的数据，频率限制在每 15 分钟 180 次或 450 次，取决于你是以用户还是应用的身份请求 API。

> 保存来自 API 资源的数据文件时，你可以保存很多文件，或者将数据写入一个文件。正如在第 6 章学到的，你也可以保存推文数据到数据库。无论你用什么方式来保存数据，确保定期保存数据，不丢失任何已经请求的数据。

在第 3 章，我们处理了一个 JSON 文件。如果在每 15 分钟之间最大化 API 使用频率，可以收集 180 个 JSON 文件。如果你碰到了频率限制问题，需要优化对 Twitter 或者其他 API 的请求，请阅读 Twitter 的“API 频率限制”（https://dev.twitter.com/rest/public/rate-limiting）这篇文章中的“避免碰到频率限制的一些建议”这一节。

13.1.3　分级数据卷

迄今为止，我们已经讨论了通过其 API 可免费获取的 Twitter 数据。但是或许你想要知道怎样能够得到所有的数据？在 Twitter 这个实例中，你可能听说过三种数据访问级别：firehose、gardenhose 和 Spritzer。Spritzer 是免费的 API。表 13-1 描述了这些级别之间的不同。

表13-1：Twitter feed类型

feed类型	覆盖	可用性	花费
firehose	所有推文	通过合作伙伴可用——DataSift（http://datasift.com/）或 Gnip（https://gnip.com/）	$$$
gardenhose	10% 的推文	新的访问不再可用	不可用
Spritzer	1% 或不到 1% 的推文	通过公共 API 可用	免费

看到这些选项你可能会想：“我需要 Firehose，因为我想要所有的数据！”但是在你尝试取得访问权限之前，需要知道下面这些事情。

- firehose 是一个很大的数据。当处理海量数据时，你需要规模化数据处理。仅仅开始查询 firehose 提供的数据集，也会需要很多的工程师和服务器。
- firehose 需要付费——一年几十万美元。这不包括你需要消耗的基础设施的花费（即服务器空间和数据库耗费）。使用 firehose 通常不是一个人可以独自做的事情——通常，由一个大型的公司或者机构支撑这些花销。

- 大部分你真正需要的，可以从 Spritzer 得到。

我们会使用 Spritzer，这是 Twitter 的免费公开 API，通过它可以在频率限制下取得推文。为了访问这个 API，我们会使用 API key 和 token。

13.1.4 API key和token

API key 和 token 是用来鉴别应用和用户的方式。Twitter API key 和 token 可能会让你迷惑。这里有四个你需要了解的概念。

- API key
  标识应用。
- API secret
  类似于应用的密码。
- token
  标识用户。
- token secret
  类似于用户的密码。

这几部分的组合给予我们访问 Twitter API 数据的权限。然而并不是所有的 API 都拥有这两层标识和密钥。Twitter 是一个很好的“最佳案例”（即更加安全的）示例。在一些情况下，API 会没有 key 或者只有一个 key。

创建一个Twitter API key和访问token

继续童工雇用的研究，收集 Twitter 上关于童工雇用的讨论。创建一个 Twitter API 的 key 很简单，但是需要以下几步。

(1) 如果你没有 Twitter 账户，先注册（https://twitter.com/signup）。
(2) 登录 apps.twitter.com。
(3) 点击“创建新应用”（Create New App）按钮。
(4) 给你的应用一个名称和描述。例如，名称设置为“童工讨论”，描述设置为“拉取 Twitter 上关于童工的讨论”。
(5) 给你的应用添加一个网站——这个网站托管着应用。指引这样写道：“如果你还没有一个 URL，可以先在此放置一个占位符，但是记得之后修改它。”我们并没有这样一个网站，所以把 Twitter 的 URL 放在这一栏中。确保你的 URL 中包含了 https，例如：https://twitter.com。
(6) 同意开发者协议，点击“创建 Twitter 应用”（Create Twitter Application）。

在创建了应用之后，你会被带到应用管理页。如果你找不到这个页面，可以通过回到应用起始页（https://apps.twitter.com/）找到它 。

现在，你需要创建一个 token。

(1) 点击“Keys 和访问 Tokens”（Keys and Access Tokens）键。（这里你可以重置 key，同样可以创建一个访问 token。）

(2) 滚动到页面底部，点击“创建我的访问 token”（Create my access token）按钮。一旦你完成了这些，这个页面会通过在顶部更新来刷新。如果再一次滚动到页面底部，你会看到访问 token。

现在你应该有了一个消费者（API）key 和一个 token。下面是我们的 key 和 token。

- 消费者 key：`5Hqg6JTZ0cC89hUThySd5yZcL`
- 消费者 secret：`Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c`
- 访问 token：`3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w`
- 访问 token secret：`nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C`

不要跟任何人分享你的 key 或 token！如果你和朋友分享了 key，他们可以在电子层面上代表你。如果他们滥用这个系统，你可能会失去访问权限，并为他们的行为负责。

为什么发布自己的应用？其中一个原因是我们可以生成更多个。在生成新的 key 和 token 的过程中，已经禁用了本书包含的 key 和 token——如果意外地暴露了 key 或者 token，这也是你应该做的事情。如果你需要创建一个新的 key，去“Keys 和访问 Tokens”（Keys and Access Tokens）栏目，点击“重新生成”（Regenerate）按钮。这会为你重新生成一个新的 API key 和 token。

现在有了一个 key，让我们访问 API！

13.2　一次简单的Twitter REST API数据拉取

有了一系列的 key 值，现在可以开始访问 Twitter 的 API 数据了。在这一节中，我们会编写一个简单的脚本，使用一个搜索查询，从 API 拉取数据。这一节中的脚本基于一个由 Twitter 提供的、作为示例的 Python 代码片段（https://dev.twitter.com/oauth/overview/single-user#python）。这份代码使用了 Python OAuth2，OAuth2 是在使用 API 时为了安全地识别和连接而使用的协议。

当下最好的认证方式是使用 OAuth2。一些 API 可能仍旧在使用 OAuth1。OAuth1 与 OAuth2 在功能上有所不同，并且是一个已经废弃的协议。如果需要使用 OAuth1，你可以使用 Requests-OAuthlib（https://requests-oauthlib.readthedocs.org/en/latest/），同 `requests` 一起拉取数据。当通过 API 认证时，确保识别哪一个协议正在被使用。如果使用了错误的协议，在尝试连接时，你会收到错误信息。

首先，需要安装 Python OAuth2：

```
pip install oauth2
```

打开一个新文件，导入 `oauth2`，并且为你的 key 变量赋值：

```
import oauth2

API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
```

```
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'
```

然后添加函数来创建 OAuth 连接：

```
def oauth_req(url, key, secret, http_method="GET", post_body="",
          http_headers=None):
    consumer = oauth2.Consumer(key=API_KEY, secret=API_SECRET)       ❶
    token = oauth2.Token(key=key, secret=secret)                     ❷
    client = oauth2.Client(consumer, token)                          ❸
    resp, content = client.request(url, method=http_method,          ❹
                           body=post_body, headers=http_headers)
    return content                                                   ❺
```

❶ 创建一个 oauth2 对象的消费者。消费者是 key 的所有者。这行代码给消费者提供 key，这样消费者可以顺利地通过 API 识别。
❷ 将 token 赋值给 oauth2 对象。
❸ 创建客户端，包含消费者和 token。
❹ 使用函数参数 url，通过 OAuth2 客户端执行请求。
❺ 返回从连接接收到的内容。

现在有了一个函数，允许我们连接到 Twitter API。然而，我们需要定义 URL，并且调用函数。搜索 API 文档（https://dev.twitter.com/rest/public/search）告诉我们更多有关想要使用的请求的信息。使用 Web 接口，可以看到，如果搜索 #childlabor，最终得到的 URL 是：https://twitter.com/search?q=%23childlabor。文档建议重新格式化 URL，所以最终的 URL 如下：https://api.twitter.com/1.1/search/tweets.json?q=%23childlabor。

之后，可以把这个 URL 作为一个变量，并使用之前定义的变量调用函数：

```
url = 'https://api.twitter.com/1.1/search/tweets.json?q=%23childlabor'
data = oauth_req(url, TOKEN_KEY, TOKEN_SECRET)

print(data)                                                ❶
```

❶ 在最后添加打印语句，这样可以看见输出。

运行脚本时，你应该看到数据打印成一个很长的 JSON 对象。你可能记得 JSON 对象看起来和 Python 字典类似，但是如果使用 print(type(data)) 重新运行脚本，你会发现内容是一个字符串。现在我们可以做以下两件事情中的一件：转化数据为一个字典并开始解析它，或者保存字符串到一个文件，之后再解析。为了继续在脚本中解析数据，在脚本顶部添加 import json。之后，在尾部，使用 json 加载字符串，并且输出它。

```
data = json.loads(data)
print(type(data))
```

变量 data 现在会返回一个 Python 字典。如果你想要将数据写入一个文件并且在之后解析它，替换为下面的代码：

```
with open('tweet_data.json', 'wb') as data_file:
    data_file.write(data)
```

最后的脚本应该看起来像下面这样：

```
import oauth2

API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'


def oauth_req(url, key, secret, http_method="GET", post_body="",
              http_headers=None):
    consumer = oauth2.Consumer(key=API_KEY, secret=API_SECRET)
    token = oauth2.Token(key=key, secret=secret)
    client = oauth2.Client(consumer, token)
    resp, content = client.request(url, method=http_method,
                                   body=post_body, headers=http_headers)
    return content


url = 'https://api.twitter.com/1.1/search/tweets.json?q=%23popeindc'
data = oauth_req(url, TOKEN_KEY, TOKEN_SECRET)

with open("data/hashchildlabor.json", "w") as data_file:
    data_file.write(data)
```

从这里开始，你可以查看 3.2 节，来解析数据。

13.3　使用Twitter REST API进行高级数据收集

从 Twitter 拉取单个数据文件并不是非常有用，因为这只返回大约 15 条推文。我们希望执行一连串的查询，这样可以收集尽可能多的关于这一话题的推文。我们会使用另外一个库来做这项工作——Tweepy。Tweepy 可以管理一系列的请求，包括 Twitter 的 OAuth。首先安装 tweepy：

```
pip install tweepy
```

在脚本的最开始，导入 tweepy，并且再一次设置你的 key：

```
import tweepy

API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'
```

之后将你的 API key 和 API secret 传入 tweepy 的 OAuthHandler 对象，这个对象会管理上一个实际提到的 OAuth 协议。之后设置你的访问 token。

```
auth = tweepy.OAuthHandler(API_KEY, API_SECRET)            ❶
auth.set_access_token(TOKEN_KEY, TOKEN_SECRET)             ❷
```

❶ 创建一个对象，通过 tweepy 来管理 API 认证。

❷ 设置访问 token。

之后，将刚刚创建的认证对象传递给 tweepy.API：

```
api = tweepy.API(auth)
```

tweepy.API 对象可以接受不同的参数，这给了你请求数据时控制 tweepy 行为的能力。你可以通过传递参数（像 retry_count=3，retry_delay=5）直接添加重试和延迟。另一个有用的选项是 wait_on_rate_limit，这个选项会直到频率限制解除后再去做下一次请求。tweepy 文档（http://docs.tweepy.org/en/latest/api.html）中有这些选项的细节和更多信息。

我们想要使用 tweepy.Cursor 创建一个和 Twitter API 的连接。然后将 API 方法（这里是 api.search，http://docs.tweepy.org/en/latest/api.html#API.search）和与其相关的参数传递给指针（cursor）。

```
query = '#childlabor'                                    ❶
cursor = tweepy.Cursor(api.search, q=query, lang="en")   ❷
```

❶ 创建 query 变量。
❷ 使用 query 创建 cursor，并且限制其只检索英语。

> 尽管 Cursor 并不是很直观，但是这是一个在数据库连接中很常用的编程名词。虽然 API 不是数据库，但类名称 Cursor 可能受 API 类似数据库使用方式的影响而命名。你可以在维基百科（https://en.wikipedia.org/wiki/Cursor_(databases)）上阅读更多关于指针的内容。

根据 tweepy 的文档（http://tweepy.readthedocs.org/en/latest/api.html），cursor 可以返回一个在单个对象级别或单页对象级别上的迭代器。你同样可以定义限制（http://tweepy.readthedocs.io/en/latest/cursor_tutorial.html#limits），来确定 cursor 抓取的页面数或对象数。如果查看 print(dir(cursor))，你会看到这里有 3 个方法：['items', 'iterator', 'pages']。一页返回一串对象，即在你的查询下独立的推文。根据需要，我们会使用页面。

让我们遍历这些页面，并且保存数据。在此之前，需要做以下两件事。

(1) 添加 import json 到脚本的顶端。
(2) 在脚本的相同目录下，创建一个名叫 data 的文件夹。为此，在命令行中运行 mkdir data。

一旦你完成了这两件事情，运行下面的代码来遍历和保存推文：

```
for page in cursor.pages():                              ❶
    tweets = []                                          ❷
    for item in page:                                    ❸
        tweets.append(item._json)                        ❹

with open('data/hashchildlabor.json', 'wb') as outfile: ❺
    json.dump(tweets, outfile)
```

❶ 对于每一个 cursor.pages() 返回的页面……
❷ 创建一个空列表来保存推文。
❸ 对于页面中的每一个对象（或推文）……

❹ 抽取 JSON 推文数据，保存到推文列表中。
❺ 打开一个名为 hashchildlabor.json 的文件，保存这些推文。

你会注意到，没有保存太多的推文到文件。每个页面只有 15 个推文，所以我们需要找出一个方法来得到更多的数据。有以下一些选项。

- 打开一个文件，并且永远不关闭它，或者打开一个文件，在末尾追加信息。这会创建一个非常大的文件。
- 将每一页保存到自己的文件中（你可以使用时间戳来保证每个文件有不同的文件名）。
- 在你的数据库中创建一个新的表来保存数据。

创建一个文件是危险的，因为进程在任何时候都可能失败，破坏数据。除非你只是在进行很小规模的数据拉取（例如，1000 条推文），或进行开发测试，否则你应该使用其他选择。

每次都有很多种方法可以将数据保存到新文件中，最普遍的一种方式是使用日期和时间戳（https://docs.python.org/2/library/datetime.html）创建一个文件，或者只是递增一个数字，将其添加到文件名的末尾。

我们会继续添加推文到简单的数据库。为此，使用下面的函数。

```
def store_tweet(item):
    db = dataset.connect('sqlite:///data_wrangling.db')
    table = db['tweets']                                  ❶
    item_json = item._json.copy()
    for k, v in item_json.items():
        if isinstance(v, dict):                           ❷
            item_json[k] = str(v)
    table.insert(item_json)                               ❸
```

❶ 创建或访问一个新表，名为 tweets。
❷ 检查推文中是否含有字典对象。由于 SQLite 并不支持保存 Python 字典，我们需要将其转换为字符串。
❸ 插入合法的 JSON 对象。

我们还需要添加 dataset 到 import 部分的代码中。在之前保存页面的地方，需要添加这个函数的使用。确保遍历每一条推文。最后的脚本应该看起来像下面一样。

```
import json
import tweepy
import dataset

API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'

def store_tweet(item):
    db = dataset.connect('sqlite:///data_wrangling.db')
    table = db['tweets']
    item_json = item._json.copy()
    for k, v in item_json.items():
        if isinstance(v, dict):
```

```
            item_json[k] = str(v)
        table.insert(item_json)

auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(TOKEN_KEY, TOKEN_SECRET)

api = tweepy.API(auth)

query = '#childlabor'
cursor = tweepy.Cursor(api.search, q=query, lang="en")

for page in cursor.pages():
    for item in page:
        store_tweet(item)
```

13.4　使用Twitter流式API进行高级数据收集

本章前文中提到过，有两种类型的 Twitter API 可以使用：REST API 和流式 API。

流式 API 同 REST API 相比有什么差别呢？下面进行了简单的概括。

- 数据是实时的，而 REST API 只返回已经发布一段时间的推文。
- 流式 API 缺乏普遍性，但是在未来，随着更多实时数据的生成和曝光，其可用性会变得越来越高。
- 因为最新的数据很有趣，所以很多人对这部分数据很感兴趣，这意味着你可以在网络上找到很多资源和帮助。

让我们创建一个脚本来收集来自流式 API 的数据。这个脚本会使用在这一章覆盖到的所有知识。首先编写最基础的部分——输入值和 key 值。

```
from tweepy.streaming import StreamListener          ❶
from tweepy import OAuthHandler, Stream              ❷

API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'
```

❶ 导入 `StreamListener`，这会创建一个流式会话，并且监听信息。

❷ 导入之前使用过的 `OAuthHandler`，以及 `Stream`，后者是真正处理 Twitter 的流信息的类。

这个脚本中的 `import` 语句和上一个脚本中的有少许不同。这两种方式都是合法的，只是个人偏好问题。下面是这两种方式的一个快速比较。

方式 1

```
import tweepy
...
auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
```

方式 2

```
from tweepy import OAuthHandler
```

```
    ...
    auth = OAuthHandler(API_KEY, API_SECRET)
```

通常情况下，脚本中没有频繁使用库时使用第一种方式。在你有一长串代码，想要更清晰一些的时候，也适合使用第一种方式。然而，当库使用得非常多时，要将其都打印出来会变得令人厌烦；同样，如果这个库是脚本的基础，从这个库导入的模块或类对人们来说应该很明显。

现在，要创建导入的 `StreamListener` 类的子类（在第 12 章学习的概念），以覆写其中的 `on_data` 方法。为此，在新的类 `Listener` 中重新定义了这个函数。当有数据时，我们想要在终端中看到它们，所以添加 `print` 语句到函数中。

```
class Listener(StreamListener):                  ❶

    def on_data(self, data):                     ❷
        print data                               ❸
        return True                              ❹
```

❶ 创建 `StreamListener` 的子类。

❷ 定义 `on_data` 方法。

❸ 输出推文。

❹ 返回 `True`。`StreamListener` 有 `on_data` 方法，同样返回 `True`。因为我们创建了子类并重新定义了这个函数，所以必须在子类方法中重复返回这个值。

接下来，添加你的认证处理逻辑：

```
auth = OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(TOKEN_KEY, TOKEN_SECRET)
```

最后，将 `Listener` 和 `auth` 传入到 `Stream` 中，开始使用搜索词过滤。在这个案例中，我们查看 child labor（童工），因为它相对于 #childlabor 更加普遍。

```
stream = Stream(auth, Listener())          ❶
stream.filter(track=['child labor'])       ❷
```

❶ 将 `auth` 和 `Listener` 作为参数传递，创建一个流。

❷ 过滤流，只返回有 child 和 labor 存在的条目。

最后的脚本如下：

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler, Stream


API_KEY = '5Hqg6JTZ0cC89hUThySd5yZcL'
API_SECRET = 'Ncp1oi5tUPbZF19Vdp8Jp8pNHBBfPdXGFtXqoKd6Cqn87xRj0c'
TOKEN_KEY = '3272304896-ZTGUZZ6QsYKtZqXAVMLaJzR8qjrPW22iiu9ko4w'
TOKEN_SECRET = 'nsNY13aPGWdm2QcgOl0qwqs5bwLBZ1iUVS2OE34QsuR4C'


class Listener(StreamListener):
```

```python
    def on_data(self, data):
        print data
        return True

auth = OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(TOKEN_KEY, TOKEN_SECRET)

stream = Stream(auth, Listener())
stream.filter(track=['child labor'])
```

下面，你需要添加代码，通过 `on_data` 方法，像本章之前那样将推文保存到数据库、文件或其他的存储工具。

13.5 小结

能够同应用编程接口进行交互是数据处理中很重要的一部分。在这一章中，我们学习了一些 API 基础知识（见表 13-2 中的总结），并且处理了来自 Twitter API 的数据。

表13-2：API概念

概念	功能
REST API（与流式 API 相比）	返回数据，并且暴露静态的节点
流式 API（与 REST API 相比）	返回查询相关的实时数据
OAuth 和 OAuth2	给定一系列 key 值和 token 的认证
分级数据卷	数据频率限制 / 可用性的不同级别；一些需要付费
key 和 token	标识用户和应用的唯一 ID 和密钥

我们已经复用了许多已经知道的 Python 概念，并且在这一章中学习了一些新的 Python 概念。首先是 `tweepy` 的使用，一个处理同 Twitter API 交互的库。你还学习了有关认证和 OAuth 协议的知识。

作为同 API 交互的拓展，第 14 章会介绍你不在场时运行 API 脚本的知识。

第 14 章

自动化和规模化

你已经从 API 和网站爬取了大量的数据，也已经清洗和组织了数据，并且运行了统计学分析，生成了可视化报告。现在是时候让 Python 大展身手，自动化你的数据处理了。在这一章中，我们会介绍如何自动化数据分析、收集和发布。我们会学习如何创建合适的日志和警报，这样可以充分地自动化脚本，得到成功、失败以及工作中碰到的任何问题的通知。

我们还会学习使用 Python 库规模化自动化程序，帮助执行多个任务，并且监控它们的成功和失败。我们会分析一些库和辅助工具，充分地在云端规模化数据。

Python 提供了大量的自动化和规模化选择。有一些简单直接的任务可以在几乎所有的机器上自动化执行，而不需要太多的设置，同时也有一些更大型、更复杂的方式来实现自动化。我们会涉及这两者的示例，同时也会讲解作为数据处理者如何规模化数据自动化。

14.1　为什么要自动化

自动化提供了一种轻松运行脚本的方式，而不需要在本地机器上执行脚本，甚至不需要你醒着！自动化的能力意味着你可以把时间花在其他的思维密集型项目上。如果拥有一个完备的脚本来执行数据清洗，你就可以专注于研究数据，写出更好的报告。

下面是一些自动化可以帮上忙的示例。

- 每周二输出一些新的分析结果；你要编制一份报告，并将其发送给相关方。
- 其他部门或同事需要能够在没有你的指导和支持下运行报告工具或清洗工具。
- 你需要每周进行一次数据下载、清洗和发送。
- 每次用户请求新报告，报告脚本需要运行，并且在报告生成后通知用户。
- 你需要每周清洗一次数据库里错误的数据，并且将其备份到其他地方。

这其中的每一个问题都有无数的解决方案，但是有一件事是确定的：它们是非常适合自动化的任务。它们的输出和步骤非常清晰。它们有一个有限但特定的听众群体。它们有确定的时间或事件来触发行动。此外，在特定的环境下它们都是可以用脚本和程序解决的事情。

当任务清晰、定义完整，并且输出非常容易确定的时候，自动化是最容易的。不管怎样，即使输出并不总是很容易测试或预测，自动化也可以帮助完成任务的一部分，把剩余的留给你（或其他人）更细致地研究和分析。你可以想象这里的自动化类似于你在生活中自动化其他的事情。你可能有一个最喜欢的已保存的比萨订单，或者一个邮件的自动回复。如果一个任务输出明确并定期发生，那么这是一个很适合自动化的任务。

但是什么时候不应该自动化呢？以下条件预示着该任务不是一个好的自动化选择。

- 任务很少发生，并且非常复杂，自己做更好（例如，填写报税单）。
- 任务的成功输出很难确定（例如，小组讨论、社会研究或调查）。
- 任务需要与人交互来确定合适的完成方式（例如，交通导航、诗歌翻译）。
- 任务成功是当务之急。

其中的一些例子，尤其是需要人工输入的例子，适合一定程度的自动化。有些任务可以通过让机器寻找建议来做部分自动化，之后再确定这些建议是对还是错（使用人工反馈的机器学习）。其他的任务，比如少见又复杂的任务，或者是非常重要的商业任务，随着对任务的熟悉，可能最后会实现自动化或部分自动化。但是你可以看到整体的逻辑来确定什么时候自动化更好，什么时候这不是一个好的想法。

如果你不确定自动化是否适合你，可以尝试自动化一些定期运行的小任务，看它如何工作。一段时间后，你很可能会发现更合适的解决方案，而且自动化一件事的经验会使你在未来自动化更多的事情更加容易。

14.2 自动化步骤

因为自动化从一个清晰简单的目标开始，所以自动化的步骤应该同样清晰和简单。文档化下面的问题对开始自动化特别有帮助（通过列表、白板、图纸或故事板）。

- 任务何时开始？
- 这个任务是否有时间限制或最大长度？如果有的话，什么时候结束？
- 对这个任务来说，有哪些必需的输入？
- 对这个任务来说，什么是成功或部分成功？
- 如果任务失败，应该发生什么？
- 任务产生或提供什么？面向谁？以何种方式？
- 在任务结束后应该发生什么（如果有的话）？

如果能够回答其中 5 个或更多的问题，你就可以开始自动化。如果不能，在你开始自动化之前，需要做更多的研究和说明。如果要自动化之前从来没有做过的事情，或者不是经常做的事情，在执行任务时尝试文档化它，然后确定你是否能够回答上述问题。

> 如果你的项目太大或太模糊，尝试将它分解为小任务，并且自动化其中的几个。或许你的任务包括一个报告，需要下载两个数据集，进行清洗和分析，然后根据输出的不同，发送结果到不同的群组。你可以分解这项任务为一些子任务，自动化每一个步骤。如果其中任何一个子任务失败了，停止执行下面的任务，警报负责维护脚本的人员，这样可以研究它，并且在 bug 或问题解决后重新执行。

所以，自动化的基本步骤如下（注意，根据任务类型的不同，步骤会有变化）。

(1) 定义你的问题集合，将其分解为更小的工作块。
(2) 精确地描述每一个子任务的输入是什么、输入做什么以及需要什么来确认任务完成。
(3) 确定哪里可以得到输入，以及何时运行任务。
(4) 开始编码你的任务，用真实或样例数据测试。
(5) 整理你的任务和脚本，添加文档。
(6) 添加日志，聚焦于调试错误和记录成功的任务。
(7) 提交你的代码到仓库中，手动测试它。按照需要做出修改。
(8) 通过将手动任务替换为自动化任务，为自动化准备好脚本。
(9) 在任务开始自动化后，关注日志和警报。修正所有的错误和 bug。更新你的测试和文档。
(10) 为日志中的错误检查频率制订一个长期计划。

自动化的第一个步骤永远是更好地定义你的任务和子任务，让它们足够小，这样可以轻易地完成它们，且它们的失败或成功是确定的。

后面的几个步骤同我们在全书中使用的处理过程类似。你应该确定怎样开始用 Python 解决问题。搜索库或者工具，帮助你修复问题或完成请求，然后开始编码。一旦脚本开始工作，你会想要使用一些不同的可用数据集或输入测试它。在成功的测试之后，简化和文档化它。将其上传到一个远程仓库（Bitbucket 或 GitHub）中，这样随着时间的推移你可以记录修改和额外的信息。

> 一旦你完成了脚本，首先手动运行它（而不是通过自动化的方式）。当新的数据就位，或者到了运行它的时候，手动执行，持续观察程序的输出。可能会有隐藏的错误存在，也可能你需要添加额外的日志和调试信息。

根据满足需求的自动化类型，你可能创建了一个简单的定时任务，脚本按特定的时间间隔执行（本章后文会介绍关于定时任务的知识）。你可能需要稍微修改脚本，让它可以通过使用参数变量、数据库或者系统上特定的文件自主运行。当它运行时，你可能会添加它到任务队列来管理它。无论哪一种自动化合适，你的工作都还没有结束。

> 当你的脚本第一次自动化时，在它每次运行时花时间来检查它是很必要的。查看日志并监控进程。你可能会找到小 bug，然后修改它们。再一次更新所有必要的日志和文档。

在大约经过了 5 次成功或日志记录下来的失败后，你可以减少人工检查的次数。然而，在每月或者每季度使用 `grep`（http://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples/）查看日志，看一下发生了什么，仍然是一个很好的主意。如果你正在使用一个日志聚合器，你完全可以自动化这一步骤，并且让这一任务发送错误和警报报告。

自动化不是小进程，但是早早投入时间和精力是值得的。一个运行良好的自动化任务集合需要一些时间来完成，但是结果通常比那些需要从始至终关注、修改和监控的一次性脚本要好。密切关注并花一些时间正确地自动化你的脚本。之后才真正投入到手头接下来的工作当中，而不是一直将你的一部分工作与监控和管理难以驾驭的任务相关联。

14.3 什么会出错

在你的自动化程序中，有很多事情可能会出问题。其中一些非常容易更正和解释，然而其他问题更加模糊，可能根本不会有一个真正的修正。自动化中的重要一课是搞清楚哪些类型的错误和问题值得花时间和精力修复，哪些问题最好使用另外的方式解决。

以在第 12 章讨论过的错误类型为例：网络爬取中的网络错误。如果碰到了重大网络错误，你只有几个好的选择。你可以改变运行任务的机器，看是否会有性能提升（这可能会带来经济和时间上的花销，取决于你的设置）。你可以找到网络提供商，寻求支持。你可以在不同的时间运行任务，看输出是否会有不同。你可以预测问题的发生，依据预测构建脚本（即在需求之外运行脚本，预测失败百分比）。

这里有很多通过自动化运行任务时可能遇到的错误：

- 数据库连接错误导致丢失或损坏数据；
- 脚本漏洞和错误，任务没有正确完成；
- 来自网站或 API 的超时错误或者过多的请求错误；
- 边界问题，报告的数据或一部分没有保证一致，导致脚本错误；
- 服务器负载问题或其他的硬件问题；
- 时间不当，竞争条件（https://en.wikipedia.org/wiki/Race_condition，如果脚本依赖于之前其他任务的完成，竞争条件能够使数据无效）。

当然还有很多潜在的你无法预料的问题。你工作的团队越大，糟糕的文档、糟糕的理解、糟糕的团队沟通伤害自动化程序的可能性就越大。你无法预防每一个错误，但是可以尝试提供最好的团队沟通和文档。尽管如此，你同样需要接受有些时候自动化程序会失败。

为了为最后的失败准备，你会希望在问题发生时收到警报。你应该确定多少比例的错误是可接受的。不是每一种服务在 100% 的时间里都表现良好（因此存在状态页面）；然而，我们可以追求完美，并确定自动化值得付出多少时间和努力。

根据你的自动化程序和它的弱点，这里有一些方法来应对这些问题。下面是一些构建弹性自动化系统的方式。

- 以特定的时间间隔重复失败的任务。
- 确保你的代码有很多的 `try...except` 代码块，让它能够处理错误。
- 在处理到机器、数据库或 API 的连接的代码周围，构建特殊的异常代码块。
- 定期维护和监控你为自动化使用的机器。
- 使用测试数据定期测试你的任务和自动化程序，确保它们正常执行。
- 注意脚本中出现的依赖、竞争条件和 API 规则，根据这些知识编写代码。
- 使用类似 `requests` 和 `multiprocessing` 的库，让困难的问题变得更简单，并尝试理解使脚本运行困难的问题所在。

我们会使用这其中的一些技术和想法，研究如何最好地监控和自动化脚本。现在，让我们讨论一下自动化中使用的工具，使数据处理更加容易和简单，并给出一些在哪里以及如何使用这些工具的建议。

14.4　在哪里自动化

根据脚本需求的不同，决定在哪里运行它是重要的第一步。无论脚本在哪里第一次执行，你都可以将它移到别的地方，但是这可能需要重写一些代码。在一开始，你可能需要它在本地运行。在*本地*运行脚本或任务就是在你自己的计算机上运行。

*远程*执行意味着在另外的机器上运行，可能是其他地方的服务器。一旦脚本运行成功并且测试良好，你会想要移动它到远程。如果你管理或拥有服务器，或者为一个有服务器的组织工作，这样迁移脚本到服务器上会相对简单。这允许你在自己的机器（笔记本电脑或台式计算机）上工作，并且不用担心在什么时间关闭和打开它。在远程运行脚本也意味着对你的网络服务提供商来说，你不是独立的。

如果无法访问服务器，但是有一台旧的不再使用的计算机或笔记本电脑，在必要的情况下，你可以将其变成服务器。如果它运行着老旧的操作系统，你可以将它升级，以便在其上运行 Python，或者可以彻底重来，安装 Linux。

> 使用家庭计算机作为远程设备意味着它需要总是运行着，并且连接到你的家庭互联网中。如果你还想安装一个之前没有使用过的操作系统，像 Linux，这是一个学习新操作系统的简单方式，并且可以帮助你过渡到管理自己的服务器。如果你刚刚开始使用 Linux，建议选择一个流行的分发版本，例如 Ubuntu（http://www.makeuseof.com/tag/ubuntu-an-absolute-beginners-guide/）或 LinuxMint（http://linuxmint.com/）。

如果你想要管理自己的服务器，但是刚起步，不用惊慌！即使你没有管理过或帮助他人管理过服务器，云服务提供商之间日益激烈的竞争，让服务器管理越来越容易。云服务提供商让你不需要了解太多的技术知识，便可以使用新机器，运行自己的服务器。服务商 DigitalOcean 有很多关于开始使用云服务的很棒的文章，包括如何创建第一台服务器（https://www.digitalocean.com/community/tutorials/how-to-create-your-first-digitalocean-droplet-virtual-server）和运行服务器（https://www.digitalocean.com/help/getting-started/setting-up-your-server/）。

无论在本地还是远程运行脚本，都有大量的工具可以用来很好地监测和更新计算机或服务

器。你希望能够非常简单地管理和更新脚本和任务，并且可以定期运行脚本以完成任务。最后，你还希望可以轻松地配置和文档化它们。在下面的几节中，我们会介绍所有这些主题，从可以让你的脚本更加自动友好的 Python 工具开始。

14.5 自动化的特殊工具

Python 提供了许多用于自动化的特殊工具。我们会查看一些使用 Python 管理自动化程序的方法，同时也会使用其他的机器和服务器完成任务。我们还会讨论怎样使用一些内置的 Python 工具管理脚本的输入，自动化看起来需要人工输入的事情。

14.5.1 使用本地文件、参数及配置文件

根据脚本的工作情况，你可能需要存储在数据库或 API 之外的参数或输入。当有一个简单的输入或输出时，你可以使用本地文件和参数来传递数据。

1. 本地文件

使用本地文件作为输入和输出时，你需要确保脚本可以每天运行在相同的机器上，或者可以简单地与输入和输出文件一起迁移。随着脚本的开发，很可能需要同时移动并改变脚本和所用文件。

我们之前使用过本地文件，但是让我们看一下如何从更加函数式的代码的角度来使用它。这段代码给了你使用标准数据类型打开和写文件的能力，并且根据脚本的需求，其复用性和扩展性很好。

```
from csv import reader, writer


def read_local_file(file_name):
    if '.csv' in file_name: ❶
        rdr = reader(open(file_name, 'rb'))
        return rdr
    return open(file_name, 'rb') ❷


def write_local_file(file_name, data):
    with open(file_name, 'wb') as open_file: ❸
        if type(data) is list: ❹
            wr = writer(open_file)
            for line in data:
                wr.writerow(line)
        else:
            open_file.write(data) ❺
```

❶ 这行代码测试文件是否适合使用 `csv` 模块打开。如果文件以 .csv 结尾，之后可以使用 CSV 读取器来打开它。

❷ 如果没有返回 CSV 读取器，这段代码返回打开的文件。如果想要根据文件扩展类型，构建一系列不同的方式来打开并解析文件，也可以做到（例如，为 JSON 文件使用 `json` 模块，或者为 PDF 文件使用 `pdfminer`）。

❸ 这段代码使用 `with...as` 返回 `open` 函数的输出，将其赋值给 `open_file` 变量。当缩进的代码块结束时，Python 会自动地关闭文件。

❹ 如果正在处理列表，这一行代码使用 CSV 输出器输出每一行列表对象为一行数据。如果有字典，可以使用 `DictWriter` 类。

❺ 我们想要一个好的备选计划，以防数据不是列表。因此，要将原始数据写到文件中。此外，根据数据类型的不同，还可以写不同的代码。

看一个例子，我们需要文件夹中最近使用的文件，这在按照日期解析日志文件，或者查看最近的网络爬虫运行结果时很有用处。

```
import os

def get_latest(folder):
    files = [os.path.join(folder, f) for f in os.listdir(folder)] ❶
    files.sort(key=lambda x: os.path.getmtime(x), reverse=True) ❷
    return files[0] ❸
```

❶ 使用 Python 内置的 `os` 模块列出每一个文件（`listdir` 方法），之后使用 `path` 模块的 `join` 方法创建一个长字符串，表示一个完整的文件路径。这是获取文件夹中所有文件的一种简单方式，只需要传递一个字符串（文件夹路径）。

❷ 通过最后修改时间来为文件排序。因为 `files` 是一个列表，所以我们可以调用 `sort` 方法，并且给它一个键作为排序的依据。这段代码传递完整的文件路径给 `getmtime` 函数，这是 `os` 模块“获取修改时间”的方法。`reverse` 参数确保最近的文件出现在列表的顶部。

❸ 返回最近的文件。

这段代码返回最近使用的文件夹，但是如果想要返回整个以最近使用文件开始的文件列表，我们可以直接修改代码，不返回第一个索引，而是返回整个列表或一个切片。

> `os` 库有很多强有力的工具来查找、修改和改变本地文件（或者服务器的本地文件）。在 Stack Overflow 上直接搜索会返回很多答案，例如如何找到过去 7 天内唯一修改的文件，或者过去一个月中唯一修改的 .csv 文件，等等。使用本地文件，特别是当你需要的数据已经存在时（或者很容易使用 `wget` 拿到），是一个很棒的简化自动化程序的方式。

2. 配置文件

为敏感信息创建本地配置文件是必需的。正像 Twelve-Factor 应用（http://12factor.net/config）中声称的那样，在代码主干之外保存配置（例如密码、登录名、邮件地址和其他的敏感信息）是成为一个好的开发者的一部分。如果你连接到数据库，发送一封邮件，使用 API 或保存支付信息，这些敏感数据应该保存在一个配置文件中。

通常情况下，我们在仓库内的一个独立文件夹存储配置文件（例如，config/）。仓库中的所有代码都会访问这些文件，但是通过使用 .gitignore 文件（见 8.4 节），可以保持这些配置文件在版本控制之外。如果有其他开发者或服务器需要这些文件，你需要手动复制它们。

建议在仓库的 README.md 中使用一个小节，介绍特殊配置文件的位置以及如何处理它们，这样新用户和合作者知道向谁索要合适的文件。

使用文件夹而不是文件允许你在运行脚本的不同机器或环境中有不同的配置。你可能想要在测试环境中有一个配置文件，使用测试的 API key，并且在生产环境使用生产环境的配置文件。你可能想要根据脚本使用的机器使用多个数据库。你可以使用一个 .cfg 文件保存这些特殊信息，如下。

```
# 示例配置文件
[address] ❶
name = foo ❷
email = myemail@bar.com
postalcode = 10177
street = Schlangestr. 4
city = Berlin
telephone = 015745738292950383

[auth_login]
user = test@mysite.com
pass = goodpassword

[db]
name = my_awesome_db
user = script_user
password = 7CH+89053FJKwjker)
host = my.host.io

[email]
user = script.email@gmail.com
password = 788Fksjelwi&
```

❶ 每一小节用一对方括号和放置于方括号中间的一个容易阅读的字符串表示。

❷ 每一行包含一个 *key = value* 的序对。ConfigParser 将这些序对作为字符串解释。值可以包含任何的字符，包括特殊字符，但是键必需遵循 PEP-8 易于阅读的语法和结构。

在配置中有小节、键和值，这允许我们使用小节的名字和键访问配置的值。这增加了 Python 脚本的清晰度，而不会变得晦涩。一旦你有了一个类似前面示例中创建的配置文件，使用 Python 解析以及在脚本中使用和自动化就很简单了。下面是一个示例。

```
import ConfigParser
from some_api import get_client ❶


def get_config(env):
    config = ConfigParser.ConfigParser() ❷
    if env == 'PROD':
        return config.read(['config/production.cfg']) ❸
    elif env == 'TEST':
        return config.read(['config/test.cfg'])
    return config.read(['config/development.cfg']) ❹
```

```
def api_login():
    config = get_config('PROD') ❺
    my_client = get_client(config.get('api_login', 'user'),
                           config.get('api_login', 'auth_key')) ❻
    return my_client
```

❶ 这是一个导入 API 客户端钩子的示例。

❷ 这段代码通过调用 ConfigParser 类初始化一个配置对象。现在这是一个空的配置对象。

❸ 这行代码调用配置解析器对象的 read 方法，并且传递一个配置文件的列表给这个函数。这里，在项目根目录下名为 config 的文件夹下保存所有的文件。

❹ 如果传递的环境变量不匹配生产环境或测试环境，则永远返回开发配置。在配置代码中做类似的判断是一个很好的做法，以防止定义环境变量失败的情况。

❺ 我们假设示例需要生产环境的 API，所以这行代码试图获取 PROD 配置。你同样可以保存这些不同环境的类型到 bash 环境中，并且使用内置的 os.environ 方法（https://docs.python.org/2/library/os.html#process-parameters）读取它们。

❻ 这行代码调用小节名称和键名称访问存储在配置中的值。这会将值作为字符串返回，所以如果你需要整数或者其他的类型，你需要转换它们。

内置的 ConfigParser 库使我们能够简单地访问配置文件中存储的小节、键和值。如果希望在不同的文件中存储不同的信息，并且为每个特定的脚本解析一个文件列表，你的代码应该类似这样：

```
config = ConfigParser.ConfigParser()
config.read(['config/email.cfg', 'config/database.cfg', 'config/staging.cfg'])
```

根据需求，由你组织代码和配置。访问配置值的语法很简单，只是使用配置中小节的名称（即 [*section_name*]）和键的名称。所以，类似下面的配置文件：

```
[email]
user = test@mydomain.org
pass = my_super_password
```

可以通过下面的方式访问：

```
email_addy = config.get('email', 'user')
email_pass = config.get('email', 'pass')
```

配置文件是一个将所有的敏感信息保存到一个地方的简单的工具。如果你更倾向于使用 .yml 或其他扩展类型的文件，Python 也有这些文件类型的读取器。确保使用某种手段将认证、敏感信息同代码分开存储。

3. 命令行参数

Python 让我们能够在自动化程序中传递命令行参数。这些参数传递关于脚本应该如何运行的信息。举个例子，如果需要脚本知道我们希望它使用开发配置运行，可以这样做：

```
python my_script.py DEV
```

我们正在使用相同的语法在命令行中运行一个文件，调用 python，之后是脚本名称，然后在这一行的最后添加 DEV。怎样才能使用 Python 解析额外的参数呢？让我们编写下面的代码：

```
from import_config import get_config
import sys


def main(env):
    config = get_config(env)
    print config


if __name__ == '__main__':
    if len(sys.argv) > 1: ❶
        env = sys.argv(1) ❷
    else:
        env = 'TEST'
    main(env) ❸
```

❶ 内置的 sys 模块，帮助完成系统任务，包括解析命令行参数。如果命令行参数列表的长度大于 1，这里存在额外的参数。第一个参数永远保存着脚本的名称（所以如果参数长度为 1，脚本名称是唯一的参数）。

❷ 为了得到参数的值，传递参数的索引到 sys 模块的 argv 方法。这行代码设置 env 为参数的值。记住，argv 方法的索引 0 永远是 Python 的脚本名称，所以从索引 1 开始解析参数。

❸ 这行代码使用解析后的参数，根据命令行参数修改代码。

如果想要解析多个额外变量，可以测试参数的长度，来确保我们有足够的参数，之后再继续解析。你可以将任何数量的参数组合到一个字符串中，但是我们建议保持在 4 个以下。如果你需要 4 个以上的参数，考虑编写一些逻辑到脚本中（例如，在星期二只执行测试，所以如果今天是星期二，使用测试部分的代码，等等）。

参数变量在你需要重新使用相同的代码执行不同的任务或者在不同的环境下运行时很有用。或许你有一个脚本来收集或分析数据，并且希望能够切换使用的环境。你或许会像下面这样运行脚本：

```
python my_script.py DEV ANALYSIS
python my_script.py PROD COLLECTION
```

或者你可能有一个脚本需要同最近更新的文件夹进行交互，抓取最新的日志——比如从多个地方抓取日志：

```
python my_script.py DEV /var/log/apache2/
python my_script.py PROD /var/log/nginx/
```

有了命令行参数，参数变量的简单变化可以创建一个可移植和健壮的自动化程序。不是每一个脚本都需要使用这些额外的变量，但是有这样一个内置到标准库中的解决方案是很棒

的，它同时提供了灵活性。

除了这些相当简单直接的方式来解析数据，给脚本额外的信息，你也可以使用更加复杂和分布式的方式，比如基于云计算的数据和数据存储。我们会在下面查看这部分内容。

14.5.2 在数据处理中使用云

云这个名词通常用来代表一个资源共享池，例如服务器。有许多公司提供云服务——亚马逊网络服务（AWS），是最著名的云服务提供商之一。

> 云这个词经常被过度使用。如果你正在云服务器上运行代码，最好说“我正在一个服务器上执行它”，而不是“我在云上执行它”。

什么时候适合使用云？如果数据太大，不能在自己的计算机上执行，或者程序需要很长的时间执行，云都是一个很好的处理方式。将大多数想要自动化的任务放到云上执行，这样你不用担心电脑打开或关闭时脚本是否在运行。

如果选择 AWS，第一次登录后会看到许多不同的服务选择。在数据处理中你只需要几种服务（见表 14-1）。

表14-1：AWS云服务

服务	在数据处理中的目的
简单存储服务（S3）	一个简单的文件存储服务，用来备份数据文件（JSON、XML 等）
弹性计算（EC2）	一个按需分配的服务器。这是运行脚本的地方
弹性 MapReduce（EMR）	通过一个托管的 Hadoop 框架提供分布式的数据处理进程

这些是你需要熟悉的基本的 AWS 服务。还有各种各样的竞争者，包括 IBM 的 Bluemix 和沃特森开发者云（让你得以使用各种大型数据平台，包括沃特森的逻辑和自然语言处理能力）。你还可以使用 DigitalOcean 或 Rackspace，它们提供了廉价的云资源。

无论使用什么，你都需要在云服务器上部署代码。为此，我们建议使用 Git（https://git-scm.com）。

使用Git部署Python

如果想让自动化程序运行在本地机器之外的地方，你需要部署 Python 脚本。我们会介绍几种简单的方式，然后介绍一些稍微复杂的方式。

> 版本控制让你的团队在同一个仓库上并行工作，不会在每个人之间产生问题。Git 允许创建不同的分支，这样你或者团队中的其他人可以在特定的需求集合上工作，或者各自完成新的集成，然后将它们合并回代码主干，而不会丢失任何核心功能。它还能确保每个人有最新的代码（包括服务器和远程机器）。

部署 Python 最简单和直观的方式是使用 Git 把仓库置于版本控制之下，用 Git 部署钩子将代码移到远程机器上。首先，需要安装 Git（https://git-scm.com/book/en/v2/Getting-Started-Installing-Git）。

如果你是 Git 新手，建议你参加 GitHub 上的 Code School 入门教程（https://try.github.io/levels/1/challenges/1）或者浏览 Atlassian 上的 Git 入门教程（https://www.atlassian.com/git/tutorials/）。上手 Git 相当简单，你会很快掌握常用命令。如果你独自在仓库上工作，不用过于担心拉取远程变化的问题，但是设置一个清晰的工作日程总是好的。

一旦 Git 安装完成，在项目代码文件夹下运行下面这些命令。

```
git init . ❶
git add my_script.py ❷
git commit -a ❸
```

❶ 初始化当前工作目录为 Git 仓库的根目录。
❷ 添加 my_script.py 到仓库中。使用来自仓库的文件名或文件夹名称——不是配置文件！
❸ 将这些改变连同任何其他运行改变（`-a`）提交到仓库。

收到提示后，编写一个提交信息，简短地解释你做出的改变，这段解释应该明确并且清晰。你可能在之后需要找到哪个提交对应代码中的哪个改变。如果信息始终编写得很清晰，这会帮助你搜索和找到这些提交。这同样会帮助团队中的其他人或者合作者理解你的代码和提交。

> 习惯于使用 `git fetch` 拉取远程变化，或使用 `git pull --rebase` 命令通过新的提交更新本地仓库。之后，在代码上工作，提交工作，推送提交到活跃的分支。当适合合并分支到主干时，你可以发送一个拉取请求（https://help.github.com/articles/using-pull-requests/），让其他人检查合并的代码，之后直接合并到主干。不要忘记删除没有用处的老的分支。

创建一个 .gitignore 文件是必需的，在这个文件中列出所有想让 Git 在推送 / 拉取时忽略的文件模式，正像在 8.4 节附注栏“Git 和 .gitignore”中讨论的那样。你可以在每个文件夹下使用一个文件，也可以只在仓库的基目录下使用一个。大多数的 Python .gitignore 文件类似于这样：

```
*.pyc
*.csv
*.log
config/*
```

这个文件会阻止仓库存储编译过的 Python 文件、CSV 文件、日志文件和配置文件。根据仓库文件夹中存在的其他文件类型，你可能想要添加更多的模式。

你可以在一些站点上挂载仓库。GitHub（https://github.com/）提供了免费的公开仓库，但是没有私有仓库。如果你需要代码是私有的，Bitbucket（https://bitbucket.org/）有免费的私有仓库。如果你已经开始在本地使用 Git，推送已有的 Git 仓库到 GitHub（https://help.github.com/articles/set-up-git/）或 Bitbucket（http://bit.ly/create_bitbucket_repo）是很简单的。

一旦创建了仓库，使用 Git 设置远程端点（服务器或服务器集群，http://git-scm.com/docs/git-remote）很简单。如果你正在部署一个使用 ssh 访问的目录，下面是一个示例。

```
git remote add deploy ssh://user@342.165.22.33/home/user/my_script
```

在推送代码到服务器之前，你需要以一些命令设置接收端的文件夹。在计划进行开发的服务器文件夹下运行这些命令：

```
git init .
git config core.worktree `pwd`
git config receive.denycurrentbranch ignore
```

这里初始化了一个空的仓库来从本地机器发送代码，同时定义了一些简单的配置，这样 Git 知道它是一个远程端点。你还需要创建一个推送–接收钩子。通过在刚刚初始化的文件夹 .git/hooks 下创建一个名为 post-receive 的可执行文件（通过许可），你可以创建一个钩子。这个文件会在部署端点接收到任何的 Git 推送后执行。它应该包含你需要在每一次推送后执行的所有任务，例如同步数据库、清理缓存或者重启任何的进程。至少，它会需要更新端点。

一个简单的 .git/hooks/post-receive 文件看起来类似于这样：

```
#!/bin/sh
git checkout -f
git reset --hard
```

这会重置所有本地的变化（在远程机器上）并且更新代码。

你需要在本地机器上做出所有的修改，测试它们，之后推送它们到部署端点。从一开始就使用这种方式是好的习惯。通过这种方式，所有代码都在版本控制之下，你可以确保没有由于直接在服务器上修改代码而导致的断断续续的 bug 或错误。

一旦远程端点设置完成，你可以直接在本地仓库运行下面的命令，使用所有最新的提交更新服务器上的代码：

```
git push deploy master
```

这样做是一个非常棒的管理仓库和服务器或远程机器的方式；很容易使用和设置，也让迁移（如果需要的话）变得直观。

如果你刚开始接触部署和版本控制，建议你从 Git 开始，熟悉它之后再转而使用更复杂的部署选择，比如 Fabric（http://www.fabfile.org/）。在本章的后面，我们会介绍一些大规模的自动化工具，用于在多个服务器之间部署和管理代码。

14.5.3　使用并行处理

并行处理对脚本自动化来说是一个很棒的工具，让你可以在一个脚本中运行多个并发进程。如果脚本需要多个进程，Python 内置的 `multiprocessing` 库会成为你自动化的得力工具。如果你有一系列的任务需要并行执行，或者通过并行化可以加速执行任务，多重处理（multiprocessing）是合适的工具。

如何使用 multiprocessing 呢？下面是一个简单的示例：

```
from multiprocessing import Process, Manager ❶
import requests

ALL_URLS = ['google.com', 'bing.com', 'yahoo.com',
            'twitter.com', 'facebook.com', 'github.com',
            'python.org', 'myreallyneatsiteyoushouldread.com']


def is_up_or_not(url, is_up, lock): ❷
    resp = requests.get('http://www.isup.me/%s' % url) ❸
    if 'is up.' in resp.content: ❹
        is_up.append(url)
    else:
        with lock: ❺
            print 'HOLY CRAP %s is down!!!!!' % url


def get_procs(is_up, lock): ❻
    procs = []
    for url in ALL_URLS:
        procs.append(Process(target=is_up_or_not,
                             args=(url, is_up, lock))) ❼
    return procs


def main():
    manager = Manager() ❽
    is_up = manager.list() ❾
    lock = manager.Lock() ❿
    for p in get_procs(is_up, lock): ⓫
        p.start()
        p.join()
    print is_up

if __name__ == '__main__':
    main()
```

❶ 从内置的 `multiprocessing` 库导入 `Process` 和 `Manager` 类，来帮助我们管理进程。

❷ 定义主要 worker 函数 `is_up_or_not`，培训需要 3 个参数：一个 URL、一个共享列表和一个共享锁。其中的列表和锁是在所有的进程间共享的，让其中的每一个进程能够修改或使用它们。

❸ 使用 `requests` 检查 `isup.me`，判定给定的 URL 当前是否在线并且可用。

❹ 测试是否可以在页面中找到文本“is up.”。如果文本存在，这个 URL 就是我们想要的。

❺ 通过 `with` 代码块调用锁的 `acquire` 方法。这会得到锁，继续执行下面缩进的代码，然后在代码块的最后释放锁。锁（http://www.laurentluce.com/posts/python-threads-synchronization-locks-rlocks-semaphores-conditions-events-and-queues/）是阻塞的，并且只能在代码中需要阻塞时使用（举个例子，如果你需要确保只有一个进程运行一组特殊的逻辑，比如检查一个共享值是否变化，或是否到达了一个终止点）。

❻ 当生成进程时，传递共享的锁和列表到函数中使用。

❼ 通过传递关键参数创建一个进程对象：目标（即，我应该执行哪个函数）和参数（即，使用什么参数）。这行代码追加所有的进程到一个列表，这样我们可以在一个地方管理它们。

❽ 初始化 `Manager` 对象，这帮助我们管理共享的对象和进程间的日志。

❾ 创建一个共享列表对象，跟踪每个站点的状态。每一个进程都能改变这个列表。

❿ 创建一个共享的锁对象，如果一个站点中不存在“is up”，停止并且宣布它。如果这些是我们管理的所有站点，我们可能有了一块重要的业务逻辑来处理紧急情况，也因此有了“停止所有程序”的理由。

⓫ 分别开始由函数 `get_proc` 返回的每一个进程。一旦它们开始执行，`join` 方法会让 `Manager` 对象和所有的子进程通信，直到最后一个进程完成。

使用多重处理的时候，你通常会有一个管理者进程和一堆子进程。你可以传递参数给子进程，可以使用共享内存和共享变量。这使你能够确定如何利用和架构 multiprocessing。根据脚本的需要，你或许想要让管理器运行脚本中一系列的逻辑，同时使用子进程运行高延迟或长时间运行的部分代码。

> 共享锁对象（https://docs.python.org/2/library/threading.html#lock-objects）提供了同步执行多个进程的能力，同时能保护内部逻辑的特定区域。有效使用它们的一个方式是直接放置锁逻辑到 `with`（https://docs.python.org/2/library/threading.html#using-locks-conditions-and-semaphores-in-the-with-statement）语句中。

如果不确定脚本是否适合使用多重处理，可以先测试脚本中的一部分代码或者一个子任务，确定你是否能够实现并行编程的目标，或者它是否将逻辑复杂化了。有一些任务使用大规模的自动化和队列来处理会更好，我们会在本章的后面讨论。

14.5.4 使用分布式处理

除了并行处理和多重处理，还有*分布式处理*，它分发进程到多台机器上（不同于发生在一台机器上的并行处理）。当计算机可以处理时，并行处理更快，但是有些时候你需要更强大的能力。

> 分布式处理涉及多个类型的计算问题。有些工具和库可以管理分布在多台计算机上的进程，还有些工具和库可以管理跨越计算机的存储。与这些问题相关的概念包括分布式计算、MapReduce、Hadoop、HDFS、Spark、Pig 和 Hive。

在 2008 年早期，克林顿总统图书馆和国家档案局发布了希拉里·克林顿在 1993 年到 2001 年作为第一夫人的日程表。这份文档包含 17 000 多页的 PDF 图片，为了将其转换为有用的数据集，需要进行光学字符识别，或者使用 OCR 技术。由于正值民主党总统候选人初选阶段，新闻组织想要发布这份数据。为了完成这项工作，《华盛顿邮报》使用了分布式进程服务，将 17 000 多张图片转化为文本。通过分发工作到 100 多台计算机上，他们不到 24 小时就完成了这项工作。

使用类似 Hadoop 的框架的分布式处理包含两个主要的步骤。第一步是*映射*数据或输入。

这一进程像是一个过滤器。使用映射器说“分离文本文件中所有的单词”或者“分离过去一个小时里推文中包含特定标签的所有用户”。下一步是 `reduce`（规约）映射后的数据为可用的形式。这类似于在第 9 章使用的聚合函数。如果我们查看所有来自 Spritzer feed 的 Twitter 句柄，可能想要根据不同的地理信息或主题统计每一个句柄的推文数量，或者所有句柄的聚合（即，所有来自这一时区的使用这些单词最多的推文）。规约部分帮助我们处理这些大数据，“规约”它为可阅读和可操作的报告。

正如你可能看到的，不是所有的数据集都需要 map-reduce，而且 MapReduce 背后的理论已经在许多 Python 数据库中可用。无论怎样，如果你有一个非常大的数据集，使用类似 Hadoop 的 MapReduce 工具会节省大量的计算时间。如果你需要一个很棒的教程，建议你阅读 Micheal Noll 写的关于用 Python 编写 Hadoop MapReduce 程序的指南（http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/），它使用一个单词计数程序来探索 Python 和 Hadoop。还有很棒的关于 mrjob（https://pythonhosted.org/mrjob/）的文档，由在 Yelp（http://www.yelp.com）的开发者编写和维护。如果你想要阅读更多关于这个主题的信息，可以查看 Kevin Schmidt 和 Christopher Phillips 的图书 *Programming Elastic MapReduce*（O’Reilly）。

如果数据集很大，但是分离存储或是实时的（或接近实时），你可能想要了解一下另外一个 Apache 项目——Spark（http://spark.apache.org/）。Spark 因为它的速度、机器学习的使用和处理流的能力获得了流行。如果你的任务处理实时的流数据（来自服务、API，甚至是日志），那么相对于 Hadoop，Spark 会是一个更灵活的选择，它可以处理相同的 MapResuce 计算结构。在你需要使用机器学习或其他需要生成数据并且“喂”给数据集群的数据分析时，Spark 也很有效。PySpark（http://spark.apache.org/docs/latest/api/python/）是 Spark 的 Python API 库，由相同的开发者维护，让你能够在 Spark 进程中编写 Python。

为了开始使用 Spark，建议你阅读 Benjamin Bengfort 的详细的博客文章（https://districtdatalabs.silvrback.com/getting-started-with-spark-in-python），其内容包括如何安装 Spark，同 Jupyter notebook 集成，以及创建第一个项目。你同样可以查看 John Ramey 关于 PySpark 和 Jupyter notebook 集成的文章（http://ramhiser.com/2015/02/01/configuring-ipython-notebook-support-for-pyspark/），在你的 notebook 中进一步探索数据收集和分析的可能性。

14.6　简单的自动化

在 Python 中，简单的自动化很容易。如果你的代码不需要在多台机器上运行，如果你拥有一台服务器，或者你的任务不是事件驱动的（或者可以每天在相同时间执行），简单的自动化会很有效。开发的一个主要原则是选择最清晰和简单的方法。自动化也是如此！如果你可以轻松地使用一个定时任务（corn job）自动化工作，绝不要浪费时间过度工程化它，或者让它变得更加复杂。

在学习简单自动化的过程中，我们会介绍内置的 cron（一个基于 Unix 系统的任务管理器）和各种 Web 接口，让团队得以轻松访问所编写的脚本。这些代表着简单的自动化解决方案，不需要你的直接干预。

14.6.1 CronJobs

Cron（http://en.wikipedia.org/wiki/Cron）是一个基于 Unix 系统的任务调度工具，用来使用服务器日志和管理工具运行脚本。Cron 需要你确定任务执行的频率和时间。

> 如果你不能轻松地为脚本确定一个时间线，cron 可能不是一个好的选择。另外，你可以运行一个规律的 cron 任务来测试是否存在运行任务所必需的条件，然后使用一个数据库或本地文件通知运行的时间。在多个定时任务的帮助下，你可以检查文件或数据库，并且执行任务。

如果你之前从来没有使用过 cron 文件，它们非常直观。大多数的任务可以通过输入下面的指令来编辑：

```
crontab -e
```

> 取决于操作系统，如果之前从来没有编写过 cron 文件，你可能会收到提示，让你选择一个编辑器。你可以使用默认的配置，或者根据其他的偏好改变它。

你会在文件中看到一些文档和注释，解释 cron 文件是如何工作的。cron 文件中每一个不以 # 符号开头的行，都定义了一个 cron 任务。每一个 cron 任务都需要下面这个参数列表：

```
minute hour day_of_month month day_of_week usercommand
```

如果脚本需要在一天中的每个小时都执行，但是只在工作日执行，你需要写类似于下面的代码：

```
0 * * * 1-5 python run_this.py
```

这告诉 cron 在周一到周五的每个整点开始执行脚本。有很多好的入门教程（https://help.ubuntu.com/community/CronHowto）详细地讲解了哪些选项对你可用，但是下面有几条建议。

- 总是在所有代码行前添加 `MAIL_TO=your@email.com` 变量。通过这种方式，如果有哪个脚本失败了，cron 会给你发送邮件，报告异常，这样你会知道它不再工作了。你需要设置笔记本电脑、台式计算机或服务器来发送邮件。根据操作系统和互联网服务提供商的不同，你可能需要做一些设置。有一个很好的 GitHub gist（https://gist.github.com/kany/c44c077881047ead8faa）可以帮助 Mac 用户上手，还有一篇来自 HolaRails 的有用的面向 Ubuntu 用户的文章（https://holarails.wordpress.com/2013/11/17/configure-sendmail-in-ubuntu-12-04-and-make-it-fast/）。
- 如果所运行的服务当计算机重启时需要重新启动，使用 `@reboot` 特性。
- 如果必需运行一些路径环境或者其他的命令来恰当地执行脚本，你应该在仓库中编写一个 cron.sh 文件。将所有必需的命令整理到一个文件中，并且直接运行这个文件，而不是使用一个用 `&&` 连接的很长的命令列表。

- 不要害怕去搜索答案。如果你是第一次使用 cron，并且遇到了问题，很有可能已经有人发布过解决方案，通过 Google 搜索就可以解决问题。

为了测试如何使用 cron，我们会创建一个简单的 Python 示例。从创建一个新的名为 hello_time.py 的 Python 文件开始，并且把这些代码放在文件里：

```
from datetime import datetime

print 'Hello, it is now %s.' % datetime.now().strftime('%d-%m-%Y %H:%M:%S')
```

下面，在相同文件夹下创建一个简单的 cron.sh 文件，并且在里面编写下面的 bash 命令：

```
export ENV=PROD
cd /home/your_home/folder_name
python hello_time.py
```

我们不需要设置环境变量，因为并不经常使用它，并且你需要更新 `cd` 这行代码，让它正确地变成代码所在的文件夹（这是指向当前文件的路径）。无论如何，这是一个很好的实例，展示了如何使用 bash 命令设置变量，查看虚拟环境变量，复制和移动文件或改变目录到新的文件夹，然后调用 Python 文件。从本书的开始，你就在使用 bash 了，所以即使你仍然是初学者也不要害怕。

最后，让我们使用 `crontab -e` 创建 cron 任务。使用编辑器在文档下方添加这些代码：

```
MAIL_TO=youremail@yourdomain.com
*/5 * * * * bash /home/your_home/folder_name/cron.sh > /var/log/my_cron.log 2>&1
```

你需要使用真实地址替换这个示例中编造的 Email 地址，编写到刚刚创建的 cron 文件的正确路径。记住，hello_time.py 脚本应该在相同的文件夹中。在这个例子里，我们还创建了一个 cron 使用的日志文件（/var/log/my_cron.log）。最后的语句 `2>&1` 告诉 cron 将输出和所有错误打印到日志文件中。一旦退出编辑器，并正确地保存了 cron 文件，你应该会看见一个信息，确认新 cron 任务已经安装。等待几分钟，然后检查日志文件。你应该会看到脚本输出的信息。如果没有的话，你可以通过在系统日志（通常为 /var/log/syslog）中搜索，或在 cron 日志（通常为 /var/log/cron）中搜索 cron 错误日志信息。为了删除这个 cron 任务，再一次编辑 crontab，删除这行代码或在这行代码的开头添加 `#`，将它注释掉。

> Cron 是自动化脚本和警报的非常简单的方式。它是在 20 世纪 70 年代中期最初的 Unix 系统的开发中，由贝尔实验室设计的非常强大的工具，直到今日，仍被广泛地使用。如果可以很简单地预测自动化程序运行时间，或者只需要几个 bash 命令就可以运行，cron 是一个自动化代码的有效方式。

如果你需要为 cron 任务传递命令行参数（见 14.5.1 节），那么文件中的代码可能类似下面这样。

```
*/20 10-22 * * * python my_arg_code.py arg1 arg2 arg3
0,30 10-22 * * * python my_arg_code.py arg4 arg5 arg6
```

Cron 是一个相当灵活也很简单的工具。如果它符合你的需要，那很棒！如果不符合，继续阅读来学习自动化数据处理的其他简单方式。

14.6.2 Web接口

如果你需要脚本、爬虫或者报告任务按需执行，一个简单的解决方案是直接构建一个 Web 接口，人们可以登录进去并点击按钮执行任务。Python 有很多不同的网络框架供你选择，所以使用哪一个框架以及花费多少时间在 Web 接口上完全取决于你。

一个简单的方式是使用 Flask-Admin（https://flask-admin.readthedocs.org/en/v1.0.9/），这是一个基于 Flask 网络框架（http://flask.pocoo.org/）构建的管理站点。Flask 是一个微框架，这意味着它并不需要太多的代码上手。在依照快速开始指引（http://flask.pocoo.org/docs/0.10/quickstart/）创建并运行了网站后，你只需要在 Flask 应用程序中创建一个视图来执行任务。

确保任务可以在完成时用其他的方式通知用户或你（邮件、通知等），因为它不太可能及时地完成并给出一个合适的 Web 响应。同样确保在任务开始时通知用户，这样他们不会发出一连串的让任务开始运行的请求。

另外一个流行并且经常使用的 Python 框架是 Bottle（http://bottlepy.org/docs/dev/index.html）。Bottle 的用法类似于 Flask，如果用户点击了按钮（或做了其他简单的操作），Bottle 用视图来执行任务。

Python 开发者使用的一个大型 Python 网络框架是 Django（https://www.djangoproject.com/）。Django 最初被开发用来使新闻编辑室可以更简单地发布内容，它拥有一个内置的认证和数据库系统，并使用配置文件配置大多数的特性。

无论你使用什么框架，或者如何创建视图，你会想要在一些地方挂载框架，这样其他人可以请求任务。你可以使用 DigitalOcean 或者亚马逊网络服务（查看附录 G）轻松挂载自己的站点。你同样可以使用支持 Python 环境的服务提供商，比如 Heroku（https://www.heroku.com/）。如果你对这些选择感兴趣，Kenneth Reitz 非常出色地介绍了如何使用 Heroku 部署 Python 应用（https://devcenter.heroku.com/articles/getting-started-with-python#introduction）。

无论使用什么框架或微框架，你都需要考虑认证和安全问题。无论使用什么 Web 服务器，你都可以在服务器端创建它，或者探索框架给你的一些选择（包括插件或者其他支持的特性）。

14.6.3 Jupyter notebook

第 10 章介绍了如何创建 Jupyter notebook，它是另外一种分享代码的好方式，特别是对于那些不需要了解 Python，但是需要查看图表或者其他脚本输出的人来说。如果你教他们如何使用简单的命令，例如运行 notebook 中所有的单元，并在下载新的报告后终止 notebook 服务，你会发现它会节省大量的时间。

添加 Markdown 单元来解释如何使用你的共享 notebook 是一种很好的方式，可以确保所有人明白如何使用代码，并且可以更容易地继续前进，而不需要你的帮助。

如果你的脚本功能组织得很好，不需要做什么修改，直接将仓库放到 Jupyter notebook 可以导入和使用代码的地方（设置的服务器或者 notebook 的 `PYTHONPATH`（http://stackoverflow.com/questions/3402168/permanently-add-a-directory-to-pythonpath）是个好主意，这样你正在使用的模块总是可用的）。通过这种方式，你可以导入那些 `main` 函数到 notebook 中，在有人点击 notebook 的“运行所有”按钮时，所有的脚本会运行并且生成报告。

14.7 大规模自动化

如果系统需要多台机器或者服务器来处理，或者报告是关于一个分布式系统或者其他事件驱动系统的，很可能你会需要比 Web 接口、notebook 和 corn 更加健壮的工具。如果你需要一个真正的任务管理系统并且想要使用 Python，你很幸运。在这一节中，我们会介绍一个健壮的任务管理工具，Celery（http://www.celeryproject.org），它能处理大量的任务，自动化 worker（你会在下一节学习 worker 相关的知识），并且提供监控解决方案。

我们还会介绍操作自动化。如果你管理着一系列的服务器或不同需求的环境，这会很有帮助。Ansible（http://www.ansible.com）是一个非常棒的自动化工具，能够帮助完成各种任务，从数据库迁移到大规模集成部署。

还有一些 Celery 的替代方案，例如 Spotify 的 Luigi（https://github.com/spotify/luigi）。如果你正在使用 Hadoop，并且有大规模任务管理的需要（特别是长时间运行的任务，这可能成为一个痛点），它非常有用。关于操作自动化的好的替代品确实有很多。如果你只需要管理很少的服务器，对于只使用 Python 的部署，一个好的选择是 Fabric（http://www.fabfile.org/）。

对于大规模的服务器管理，一个很好的选择是 SaltStack（http://saltstack.com/），或同很多部署与管理工具［例如 Chef（https://www.chef.io/chef/）或 Puppet（https://puppetlabs.com/）］一起使用 Vagrant（https://www.vagrantup.com/）。我们选择重点介绍在这一节中使用的一些工具，但是它们不是使用 Python 处理大规模自动化程序的唯一选择。考虑到领域的声望和必要性，我们建议在你最喜欢的科技论坛上参与大规模自动化的讨论，例如 Hacker News（https://news.ycombinator.com/）。

14.7.1 Celery：基于队列的自动化

Celery（http://www.celeryproject.org/）是一个用来创建分布式队列系统的 Python 库。有了 Celery，可以通过调度器或者通过时间和消息来管理任务。如果你正在寻找能够处理长时间运行的时间驱动的任务的、可扩展的工具，Celery 是一个很完整的解决方案。Celery 很好地集成了几个不同的队列。它使用设置文件、用户接口和 API 调用来管理任务。它非常容易上手，所以如果这是你第一次使用任务管理系统的话，没有必要害怕。

无论你如何创建 Celery 项目，都很可能包含下面的任务管理系统组件。

- 消息中间件（例如 RabbitMQ，https://www.rabbitmq.com/）
  消息中间件类似于一个等待处理的任务的队列。

- 任务管理器 / 队列管理器（Celery）
  这个服务会跟踪一些逻辑，这些逻辑控制着使用多少 worker、什么任务有优先权、以

及何时重试，等等。

- worker
  worker 是通过 Celery 控制的 Python 进程，执行 Python 代码。它们知道你让它们去执行什么任务，并且尝试执行这些 Python 代码到结束。

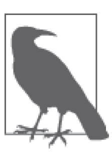
- 监控工具［例如 Flower（http://flower.readthedocs.org/en/latest/）］
  允许查看 worker 和队列，对于回答类似于“昨晚什么失败了”这样的问题很有用。

Celery 有一个很有用的上手指南（http://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html），但是我们发现最大的问题并不是学习如何使用 Celery，而是了解什么类型的任务适合于队列、什么类型不适合于队列。表 14-2 综述了一些有关基于队列自动化的问题和原则。

表14-2：队列还是非队列？

基于队列的任务管理需求	不需要队列的自动化需求
任务没有一个确定的截止日期	任务可以并且的确有截止日期
不需要知道有多少任务	可以轻松地量化需要完成什么任务
只是大体上知道任务的优先级	清楚地知道任务的优先级
任务不需要总按顺序发生，或者通常不是有顺序的	任务必需按顺序发生
任务有时花费很长的时间，有时花费很短的时间	需要知道任务花费多长时间
任务调用（或排队）基于一个事件或其他的任务结束	任务基于时间或其他可预测的事情
任务失败是可以的；可以重试	必须了解每一次任务的失败
有大量的任务，而且很有可能继续增长	每天只有几个任务

这些都是一般性的需求，但是它们指出了关于何时选择任务队列和何时最好运行在一个有警报、监控和日志的调度器上，这两者之间的一些理念区别。



任务的不同部分位于不同的系统中是没问题的，而且你在大公司会经常看到不同的任务“桶”（bucket）。也可以同时测试基于队列的和不基于队列的任务管理工具，以确定哪一种最适合你和你的项目。

Python 也有其他的任务和队列管理系统，包括 Python RQ（http://python-rq.org/）和 PyRes（https://github.com/binarydud/pyres）。这两者都是很新的库，因此在解决问题方面，可能没有足够的资源能够通过 Google 搜索到 ，但是如果你想要从 Celery 开始，之后转移到其他的选择上，你有了备选项。

14.7.2 Ansible：操作自动化

如果你当前的规模需要 Celery 帮助管理任务，很可能你同样需要一些帮助，来管理其他的服务和操作。如果项目需要在一个分布式系统上维护，你应该开始组织它们，这样可以轻松地通过自动化实现分布式。

Ansible（http://www.ansible.com/home）是一个自动化项目操作的非常棒的系统。Ansible 提供

了一系列的工具，你可以用来迅速编写、部署和管理代码。你可以使用 Ansible 来迁移项目，备份远程机器上的数据。你还可以使用它根据需要来使用安全补丁或新的包更新服务器。

Ansible 有一个快速开始的视频（http://docs.ansible.com/quickstart.html），可用来了解所有的基础知识，但是我们同样想强调文档中描述的几个最有用的特性：

- MySQL 数据库管理（http://docs.ansible.com/mysql_db_module.html）
- Digital Ocean 基本单元和键管理（http://docs.ansible.com/ansible/list_of_cloud_modules.html#digital-ocean）
- 滚动升级和部署指南（http://docs.ansible.com/guide_rolling_upgrade.html）

同样建议你查看 Justin Ellingwood 的关于 Ansible 操作的介绍（https://www.digitalocean.com/community/tutorials/how-to-create-ansible-playbooks-to-automate-system-configuration-on-ubuntu）和 Servers for Hackers 关于 Ansible 的扩展介绍（https://serversforhackers.com/an-ansible-tutorial）。

> 如果你只有一两台服务器，或者只部署一两个项目，Ansible 很可能太高级或过于复杂，但是随着项目的成长，你需要一些工具来帮助保持项目的有效组织时，它就是一个很好的资源。如果你对操作和系统管理有兴趣，这是一个很值得学习和掌握的工具。

如果你更喜欢把操作留给你创建的一个很好的镜像上，每一次操作时只需要重新启动，大量的云服务提供商允许你这么做！对于你的数据处理需求来说，你并不需要成为一个操作自动化专家。

14.8 监控自动化程序

花费时间监控自动化程序是必需的。如果你不知道任务是否完成了，或者不知道任务成功还是失败了，你最好还是不要运行它们了。因此，监控你的脚本和运行它们的机器是过程中非常重要的一部分。

举个例子，如果有一个隐藏的 bug，数据并没有被加载，而且每天或每周，你都在老的数据上运行报告，这是个非常可怕的消息。在自动化程序中，失败并不总是很明显，因为脚本可能使用旧数据或者在存在错误和不一致的情况下继续运行。监控是观察脚本成功或者失败，即使所有的迹象都表明脚本仍然在正常地执行。

> 监控可以有小的或者大的足迹，这取决于任务的规模和需求。如果你会有一个大规模的自动化程序，跨越多个服务器，你可能需要使用一个大型的分布式监控系统，或者具有监控功能的服务。然而，如果你正在一个家庭服务器上运行任务，可能只需要使用内置的 Python 日志工具。

你可能还想在脚本中使用一些警报和通知工具。在 Python 中，上传、下载、用邮件发送，甚至用 SMS 发送结果都非常简单。在这一节，我们会介绍几种不同的日志选择，并查看

发送通知的方法。在全面测试并深入理解日常监控所有潜在的错误之后，你可以充分地自动化任务，并通过警报管理错误。

14.8.1 Python日志

最基本的监控脚本的方式是日志。你很幸运，Python 有一个非常健壮并且有丰富特性的日志环境，它是标准库的一部分。与你交互的客户端或库通常都拥有集成了 Python 日志系统的日志工具。

使用 Python 内置的日志模块中提供的简单基本的配置，可以实例化日志器，并且开始记录。之后你可以使用很多不同的配置选项，来迎合脚本特殊的日志需求。Python 的日志模块允许你设置特定的日志等级（https://docs.python.org/2/library/logging.html#logging-levels）和日志的记录属性（https://docs.python.org/2/library/logging.html#logrecord-attributes），调整日志的格式。日志器对象同样拥有方法和属性（https://docs.python.org/2/library/logging.html#logger-objects），根据你的需要，这也会很有用处。

下面是在代码中如何创建并使用日志的实例。

```
import logging
from datetime import datetime


def start_logger():
    logging.basicConfig(filename='/var/log/my_script/daily_report_%s.log' %
                        datetime.strftime(datetime.now(), '%m%d%Y_%H%M%S'), ❶
                        level=logging.DEBUG, ❷
                        format='%(asctime)s %(message)s', ❸
                        datefmt='%m-%d %H:%M:%S') ❹


def main():
    start_logger()
    logging.debug("SCRIPT: I'm starting to do things!") ❺

    try:
        20 / 0
    except Exception:
        logging.exception('SCRIPT: We had a problem!') ❻
        logging.error('SCRIPT: Issue with division in the main() function') ❼

    logging.debug('SCRIPT: About to wrap things up!')

if __name__ == '__main__':
    main()
```

❶ 使用 `logging` 模块的 `basicConfig` 方法，初始化日志系统，这需要一个日志文件名称。这段代码输出日志到 /var/log 文件夹中的文件夹 my_script。文件名是 daily_report_<DATEINFO>.log，其中 <DATEINFO> 是脚本开始执行的时间，包括月、日、年、小时、分钟和秒。这告诉我们何时以及为什么脚本开始运行，同时这是一个好的日志实践。

❷ 设置日志级别。通常，你会想要把级别设置为 `DEBUG`，这样可以在代码中留下调试信息，在日志中跟踪它们。如果想要更多的信息，你可以使用 `INFO` 级别，这会展示更多的来自辅助库的日志信息。一些人更喜欢简略的日志，设置日志级别为 `WARNING` 或 `ERROR`。

❸ 使用日志的记录属性设置 Python 日志的格式。这里我们记录发送给日志的信息和打印日志的时间。

❹ 设置一个易于阅读的日期格式，这样很容易使用我们喜欢的日期格式解析或搜索日志。这里我们记录了月、日、小时、分钟和秒。

❺ 调用模块的 `debug` 方法开始记录日志。这个方法需要一个字符串。我们用单词 SCRIPT: 作为脚本日志实体的前缀。在日志中添加类似这样可搜索的标签，会在之后帮助你确定哪个进程或库写了这条日志。

❻ 使用 logging 模块的 `exception` 方法，输出你发送的字符串和 Python 异常的回溯信息，因此只能够在一个异常代码块中使用。这在调试错误和查看脚本中有多少异常时非常有用。

❼ 使用级别 `error` 打印一个长一点的错误信息。`logging` 模块能够打印不同级别的错误信息，包括 `debug`、`error`、`info` 和 `warning`。日志级别应该与打印的内容一致，为正常信息使用 `info` 或 `debug` 级别，用 `error` 打印脚本中的错误和异常信息。通过这种方式，你永远知道去哪里找问题，以及如何在检查中正确地解析日志。

正像在示例中做的那样，我们发现开始日志信息时，用一个关于正在输出信息的代码模块或区域的标签是很有用处的。这会帮助确定错误发生在哪里。这也使得日志便于搜索和解析，因为你可以清晰地看到脚本碰到了什么错误或问题。学习日志最好的方式是在第一次编写脚本的时候确定在哪里放置信息，并在脚本中保留重要的信息，以确定是否发生了问题以及在哪些节点发生了问题。

> 即使已经预料到了异常，每一个异常也应该通过日志记录下来。这会帮助你跟踪异常发生的频率，以及你的代码是否应按照正常情况处理它们。`logging` 模块提供了 `exception` 和 `error` 方法，你可以打印异常和 Python 的回溯，并且可以使用 `error` 添加一些额外的信息，详细描述可能发生了什么，以及哪部分代码触发了这个错误。

你还应该将同数据库、API 和外部系统的交互通过日志记录下来。这会帮助你确定脚本与它们的交互何时出现了问题，确保它们是稳定的、可信赖的或者可以使用的。你与之交互的许多库同样能够根据配置打印日志。例如，`requests` 模块会将连接问题和请求直接记录到你的脚本日志中。

即使不为脚本创建任何其他的监控或警报，你也应该使用日志。它很简单，并且为未来的你和其他人提供了优秀的文档。日志并不是唯一的解决方案，但是它们是一个好的标准，并且是自动化程序监控的基础。

除了日志之外，你可以为脚本设置易于分析的警报。在下一节中，我们会介绍几种使脚本能够发送有关成功和失败信息的方式。

14.8.2 添加自动化信息

发送报告、跟踪脚本、通知自己错误的一个简单方式是使用邮件或者其他直接从脚本发送的信息。有很多 Python 库能帮助完成这个任务。精确地确定脚本和项目中需要什么类型的信息是一个好的开始。

问问自己下列选项是否适用于你的脚本。

- 它会产生一个报告，并将报告发送到特定的收件人列表。
- 它有一个清晰的成功 / 失败信息。
- 它与其他的合作者或同事相关。
- 它提供了不易在网站或快速指示板上查看的结果。

如果这其中的任何一个听起来像你的项目，你的项目很可能适合某种方式的信息自动化。

1. 邮件

使用 Python 处理邮件非常直观。建议你通过你最喜欢的邮件提供商（我们使用 Gmail），构建一个独立的脚本的 email 地址。如果它没有立即自动同 Python 集成，很可能能够通过在网上搜索发现合适的配置列表和有用的示例配置。

让我们看一下曾用来发送带有附件的邮件到收件人列表的脚本。我们修改了 @dbieber 编写的代码片段（https://gist.github.com/dbieber/5146518），这段代码修改自 Rodrigo Coutinho 的博文“使用 Python 发送 Gmail 邮件”（http://kutuma.blogspot.jp/2007/08/sending-emails-via-gmail-with-python.html）：

```
#!/usr/bin/python
# Adapted from
# http://kutuma.blogspot.com/2007/08/sending-emails-via-gmail-with-python.html
# Modified again from: https://gist.github.com/dbieber/5146518
# config file(s) should contain section 'email' and parameters
# 'user' and 'password'

import smtplib ❶
from email.MIMEMultipart import MIMEMultipart ❷
from email.MIMEBase import MIMEBase
from email.MIMEText import MIMEText
from email import Encoders
import os
import ConfigParser


def get_config(env): ❸
    config = ConfigParser.ConfigParser()
    if env == "DEV":
        config.read(['config/development.cfg']) ❹
    elif env == "PROD":
        config.read(['config/production.cfg'])
    return config


def mail(to, subject, text, attach=None, config=None): ❺
```

```
    if not config:
        config = get_config("DEV") ❻
    msg = MIMEMultipart()
    msg['From'] = config.get('email', 'user') ❼
    msg['To'] = ", ".join(to) ❽
    msg['Subject'] = subject
    msg.attach(MIMEText(text))
    if attach: ❾
        part = MIMEBase('application', 'octet-stream')
        part.set_payload(open(attach, 'rb').read()) ❿
        Encoders.encode_base64(part)
        part.add_header('Content-Disposition',
                        'attachment; filename="%s"' % os.path.basename(attach))
        msg.attach(part)
    mailServer = smtplib.SMTP("smtp.gmail.com", 587) ⓫
    mailServer.ehlo()
    mailServer.starttls()
    mailServer.ehlo()
    mailServer.login(config.get('email', 'user'),
                     config.get('email', 'password'))
    mailServer.sendmail(config.get('email', 'user'), to, msg.as_string())
    mailServer.close()


def example():
    mail(['listof@mydomain.com', 'emails@mydomain.com'],
         "Automate your life: sending emails",
         "Why'd the elephant sit on the marshmallow?",
         attach="my_file.txt") ⓬
```

❶ Python 内置的 `smtplib` 库（https://docs.python.org/2/library/smtplib.html）提供了一个 SMTP 的封装，SMTP 是发送和接收邮件的标准协议。

❷ Python 的 `email` 库（https://docs.python.org/2/library/email.html）帮助创建邮件信息和附件，并用合适的形式保存它们。

❸ 函数 `get_config` 从一系列的本地配置文件中加载配置。我们传递一个环境变量，环境变量可以为字符串 `"PROD"` 或 `"DEV"`，来表明这是在本地运行（`"DEV"`）还是在远程生产环境运行（`"PROD"`）的程序。如果只使用一个环境，你可以简单地返回项目中唯一的配置文件。

❹ 这行代码使用 Python 的 `ConfigParser` 读取 .cfg 文件，返回 `config` 对象。

❺ 我们的 `mail` 函数接受一个邮件地址列表，作为变量 `to`、邮件的主题和文本、一个可选的附件和一个可选的 `config` 变量。附件需要是本地文件的名称。`config` 对象需要是 `ConfigParser` 的实例。

❻ 这行代码设置了默认配置。安全起见，我们使用 `"DEV"` 配置。

❼ 这行代码使用了 `ConfigParser` 对象来从配置文件中拉取邮件地址。这个变量安全地保存地址，并与仓库代码分离。

❽ 这段代码拆分邮件列表，并且用逗号和一个空格分离它们。这会展开邮件地址的列表为一个字符串，因为这是我们 MIME 需要的类型。

❾ 如果有附件的话，这行代码按照 MIME 分组标准开始对附件做特殊处理，以发送附件。

⑩ 这段代码使用传入的文件名字符串打开并读取完整的文件。
⑪ 如果你没有使用 Gmail，设置这些参数为邮件服务提供商 SMTP 的主机和端口。如果有好的文档的话，很容易找到这些参数。如果没有的话，直接搜索“SMTP 设置 < 你的邮件服务提供商名称 >”应该会得到答案。
⑫ 这是一些示例代码，提示你 mail 函数所需的参数。你可以看到需要的数据类型（字符串、列表、文件名）和顺序。

简单的 Python 内置库 `smtplib` 和 `email` 使用它们的类和方法帮助我们快速地创建和发送邮件信息。将脚本中的一些其他部分（例如在配置文件中保存邮件地址和密码）抽象出来是保证脚本和仓库安全可复用所必需的。一些默认的设置确保脚本永远可以发送邮件。

2. 短消息服务（SMS）和语音

如果想要集成电话信息到警报程序中，你可以使用 Python 来发送文本信息，或者打电话。Twilio（https://www.twilio.com）是一个经济有效的选择，支持多媒体信息和自动拨打电话。

> 在开始使用 Twilio API 之前，你需要注册得到认证代码和 key，并且安装 Twilio 的 Python 客户端（https://github.com/twilio/twilio-python）。在 Python 客户端的文档（https://twilio-python.readthedocs.org/en/latest/）中有一个很长的代码示例列表，所以如果你需要做一些有关语音或者文本的事情，很可能这里有一个好的可用的特性。

看一下发送一个快速的文本信息是多么简单。

```
from twilio.rest import TwilioRestClient ❶
import ConfigParser


def send_text(sender, recipient, text_message, config=None): ❷
    if not config:
        config = ConfigParser('config/development.cfg')

    client = TwilioRestClient(config.get('twilio', 'account_sid'),
                              config.get('twilio', 'auth_token')) ❸
    sms = client.sms.messages.create(body=text_message,
                                     to=recipient,
                                     from_=sender) ❹

def example():
    send_text("+11008675309", "+11088675309", "JENNY!!!!") ❺
```

❶ 我们会使用 Twilio Python 客户端通过 Python 直接同 Twilio API 交互。
❷ 这行代码定义了一个可以用来发送文本的函数。我们需要发送者和接收者的电话号码（以国家代码开头），以及想要发送的简单文本信息，我们还能传递一个配置对象。我们会使用配置来同 Twilio API 进行认证。
❸ 这段代码创建了一个客户端对象，它会使用我们的 Twilio 账户认证。当你注册 Twilio 账户时，会收到一个 `account_sid` 和一个 `auth_token`。将它们放在脚本使用的配置文件中名为 `twilio` 的小节中。

❹ 为了发送一条信息，这段代码使用了客户端的 SMS 模块，并且调用了 message 资源的 `create` 方法。正像 Twilio 文档中写的（https://twilio-python.readthedocs.io/en/latest/usage/messages.html#sending-a-text-message），之后可以通过几个参数发送一条文本信息。

❺ Twilio 在全世界范围工作，并且需要基于国际的电话号码。如果你不确定要使用什么国际电话代码，维基有一个很棒的列表（https://en.wikipedia.org/wiki/List_of_country_calling_codes）。

> 如果你对让脚本通过 Python“交谈”感兴趣，Python 的 text-to-speech 模块（https://pyttsx.readthedocs.org/en/latest/）可以在电话上“朗读”你的文字。

3. 集成聊天

如果你想要集成聊天室到警报系统中，或者你的团队或合作者普遍使用聊天室，有很多 Python 聊天室工具可用。根据聊天室客户端和需要，可以选择 Python 或者基于 API 的解决方案，你可以使用关于 REST 客户端的知识，连接并向正确的人发送消息。

如果你使用 HipChat，他们的 API（https://www.hipchat.com/docs/apiv2）很容易同 Python 应用程序或脚本集成。这里有各种各样的 Python 库（https://www.hipchat.com/docs/apiv2/libraries），可以简单地发送信息到聊天室或直接发送给个人。

为了开始使用 HipChat API，你需要首先登录（https://hipchat.com/account/api）并获取一个 API token。你可以使用 Python 库 HypChat（https://github.com/RidersDiscountCom/HypChat），发送一个快速信息到聊天室。

首先，使用 pip 安装 HypChat：

```
pip install hypchat
```

现在，使用 Python 发送一条信息！

```
from hypchat import HypChat
from utils import get_config


def get_client(config):
    client = HypChat(config.get('hipchat', 'token')) ❶
    return client


def message_room(client, room_name, message):
    try:
        room = client.get_room(room_name) ❷
        room.message(message) ❸
    except Exception as e:
        print e ❹


def main():
```

```
config = get_config('DEV') ❺
client = get_client(config)
message_room(client, 'My Favorite Room', "I'M A ROBOT!")
```

❶ 使用 HypChat 库同聊天室客户端交谈。这个库使用 HipChat token 初始化一个新的客户端，我们将 token 保存在配置文件中。

❷ 这行代码使用 `get_room` 方法，选择与字符串名称匹配的房间。

❸ 这行代码使用 `message` 方法发送一条信息到一个房间或者一个用户，传递给函数一个简单的字符串，里面包含想说的内容。

❹ 总是在基于 API 库的周围使用 `try...except` 代码块，以应对连接错误或者 API 改变带来的异常。这段代码打印了错误，但是你可能更想打印日志，以充分地自动化脚本。

❺ 这里使用的函数 `get_config` 是从不同的脚本中导入的。通过引入这些辅助函数，并且将它们放置到独立的模块中复用，我们遵循了模块化的代码设计。

如果想要登录到聊天室，你可以使用 HipLogging（https://github.com/invernizzi/hiplogging）探索这些选项。取决于需求和团队的工作方式，你可以根据自己的喜好设置聊天室的登录功能；但是好消息是，你总是可以在他们可能看见的地方留下笔记！

如果你更喜欢 Google Chat，有很多非常棒的示例展示了如何使用 SleekXMPP（http://sleekxmpp.com/getting_started/sendlogout.html）做这件事。你还可以使用 SleekXMPP 发送 Facebook 聊天信息（http://stackoverflow.com/questions/18906462/send-facebook-messages-via-sleekxmpp/21452035#21452035）。

关于 Slack 信息，查看 Slack 团队的 Python 客户端（https://github.com/slackhq/python-slackclient）。

对于其他的聊天客户端，建议你使用 Google 搜索“Python <你的客户端名称>”。很有可能有人已经尝试使用他们的 Python 代码同这个客户端连接，又或者有 API 可供你使用。在第 13 章中，你知道了如何在工作中使用一个 API。

关于脚本（和自动化）成功或失败的警报的选择有这么多，很难知道应使用哪一个。重要的是选择一个你或你的团队经常使用的方式。优先考虑易用性和与日常生活的整合是必需的，自动化会帮助你节省时间，不让你花费更多的时间来检查服务。

14.8.3　上传和其他报告

如果你需要上传报告或图片到独立的服务或文件分享作为自动化的一部分，有很多很棒的工具。如果这是一种在线的形式，或者一个需要与之交互的网站，建议使用第 12 章中的 Selenium 爬取技巧。如果这是一个 FTP 服务器，Python 有一个标准 FTP 库（https://docs.python.org/2/library/ftplib.html）。如果需要发送报告给一个 API，或者通过 Web 协议发送，你可以使用 `requests` 库，或者在第 13 章中学到的 API 技巧。如果需要发送 XML，你可以使用 LXML（查看第 11 章）。

无论准备同哪些服务交互，你很有可能与服务有过接触和交流。我们希望你自信地练习这些技能，并且独立思考。

14.8.4 日志和监控服务

如果一个脚本满足不了需求，或者你想将自动化程序合并到一个大型的组织框架中，你可能需要研究将日志和监控作为一个服务的技术。有很多公司通过创建工具和系统来跟踪日志，致力于让数据分析师和开发者的生活更简单。这些工具通常有非常简单的 Python 库，将日志和监控信息发送到它们的平台。

> 通过日志服务，你可以将更多的时间投入到研究和脚本中，将更少的时间用于管理监控和日志。这可以将“我们的脚本在不在工作，工作得怎么样？”这样的问题留给团队中的非开发者，因为其中的很多服务都有很棒的报表和内置的警报。

根据自动化程序的大小和布局，你可能同时需要系统监控以及脚本与错误监控。在这一节中，我们会查看几种能够同时完成这两件事的服务，同时也会查看一些更加专业的服务。即使你的自动化程序的规模没有大到需要它们，但是了解什么是可用的总是好的。

1. 日志和异常

基于 Python 的日志服务提供了打印日志到一个中心服务的能力，同时让你的脚本在一系列不同的机器上运行，无论是本地机器还是远程机器。

一个有着很棒的 Python 支持的类似服务是 Sentry（https://getsentry.com/welcome/）。只需要每月支付很少的费用，你就可以访问错误的报告板，收到基于异常临界值发送的警报，监控基于日、周、月的错误和异常类型。Sentry 的 Python 客户端（https://github.com/getsentry/raven-python）非常容易安装、配置和使用。如果你正在使用类似 Django、Celery 的工具，甚至是简单的 Python 日志工具（https://docs.sentry.io/clients/python/），Sentry 都有相应的集成接口，所以你不需要大量地修改代码来开始工作。最重要的是，基础代码持续地更新，工作人员非常友善，在你有问题的时候乐于帮助你。

其他选择包括 Airbrake（https://airbrake.io/languages/python_bug_tracker）和 Rollbar（https://rollbar.com）。Airbrake 最开始是一个基于 Ruby 的异常跟踪器，现在开始支持 Python。这是一个很受欢迎的市场，所以在本书开始印刷时很可能会有新的库发布。

还有拉取和解析日志的服务，例如 Loggly（https://www.loggly.com/）和 Logstash（https://www.elastic.co/products/logstash）。这些服务允许你在聚合的日志级别上监控它们，同样也可以解析、搜索以及在日志中寻找问题。这些工具只在你有足够的日志和时间检查它们时才有用，但是对于有着大量日志的分布式系统非常有用处。

2. 日志和监控

如果你有分布式机器或者正在集成脚本到公司或大学的基于 Python 的服务器环境，你可能想要更健壮的监控，不仅仅是针对 Python，而是整个系统。有很多服务提供了系统数据库负载监控、Web 应用程序的监控，同样也有自动化任务的监控。

New Relic（http://newrelic.com/）是这方面最流行的服务之一，它可以监控服务器和系统进程，也可以监控 Web 应用程序。使用 MongoDB 和 AWS ？或者 MySQL 和 Apache ？ New Relic 插件（http://newrelic.com/plugins）让你可以轻松地将服务的日志集成到用于监控服务

器和应用健康的报告板中。除此之外，它们提供了一个 Python 代理（https://docs.newrelic.com/docs/agents/python-agent/getting-started/introduction-new-relic-python），通过它你可以很容易地为 Python 应用（或脚本）在相同的生态系统中打印日志。在整合所有的监控工具之后，发现问题和创建合适的警报系统会变得更简单，这样团队中的相关人员会及时了解任何发生的问题。

另外一个系统和应用监控服务是 Datadog（https://www.datadoghq.com/）。Datadog 允许你整合许多服务（https://www.datadoghq.com/product/integrations/）到一个报告板中。这会节省时间和精力，让你轻松地发现项目、应用和脚本中的错误。Datadog 的 Python 客户端（https://github.com/DataDog/datadogpy）能够打印你想监控的不同事件的日志，但是需要一些个性化设置。

无论你使用什么监控器，也无论你是决定构建自己的监控器还是使用一个服务，拥有规律的警报、深入了解你使用的服务、理解代码和自动化系统的完整性是非常重要的。

> 当依赖自动化程序完成工作和项目的其他部分时，你应该确保监控系统容易使用并且直观，这样你可以专注于项目中更重要的部分，不用承担疏忽错误或其他问题的风险。

14.9 没有万无一失的系统

正如本章讨论的那样，完全依赖于任何系统都是鲁莽的，应该避免。无论你的脚本或系统看起来多么完美无缺，都会在一些时间点失败。如果你的脚本依赖于其他的系统，它们可能在任何时间点失败。如果你的脚本包含来自 API、服务或网站的数据，有可能 API 或网站会改变或崩溃，以致需要维修，或者可能会发生任意数量的其他事件，导致自动化失败。

如果一个任务是绝对的关键任务，就不应该自动化。你可以自动化其中的一部分或绝大部分，但是它总是需要高度的监督和工作人员来确保它没有失败。如果这很重要，但不是最重要的部分，对这一部分的监控和警报应该反映出它的重要程度。

> 随着深入数据处理和自动化，你会花费越来越少的时间构建高质量的任务和脚本，更多的时间会花费在解决问题、关键问题的思考以及应用分析领域的方法论和领域知识到工作中。自动化可以帮助你做这些事情，但是对于自动化什么重要任务以及如何自动化，持谨慎的态度总是好的。

随着自动化程序的成熟和发展，你不仅改进了自动化程序，让它有更好的伸缩性，同时增加了对代码库、Python 以及数据与报告的理解。

14.10 小结

你已经学习了使用小规模和大规模解决方案来自动化大量的数据处理工作。你可以通过日志、监控和基于云的解决方案监控和跟踪脚本和任务以及子任务，这意味着你可以花费更少的时间跟踪这些事情，花费更多的时间在真正的报告上。你已经定义了自动化可能成功

和失败的方式，并帮助创建了一套清晰的关于自动化的指南（理解所有的系统最后都会也必将失败）。你知道如何给其他的团队成员和同事权利，以后他们可以自己运行任务，同时你也学习了一些部署和设置 Python 自动化的知识。

表 14-3 总结了本章中的新概念和库。

表14-3：新的Python和编程概念和库

任务/库	目的
在远程运行脚本	在一台服务器或者其他的机器上运行代码，这样你就不必担心自己使用计算机时会受到干扰
命令行参数	在运行 Python 脚本时使用 `argv` 解析命令行参数
环境变量	使用环境变量参与实现脚本逻辑（例如运行在什么机器上，使用什么配置）
使用 Cron	在你的服务器或远程计算机上编写一个 shell 脚本来执行一个 cron 任务。是一种基本的自动化形式
配置文件	使用配置文件为脚本定义敏感或特殊的信息
Git 部署	使用 Git 轻松地部署代码到一台或更多的远程机器上
并行处理	Python 的 `multiprocessing` 模块让你能在同一时间轻松地运行很多进程，同时有共享数据和锁机制
MapReduce	有了分布式数据，你可以根据特定的属性映射数据，或者通过执行一系列的任务，之后规约数据，聚合分析
Hadoop 和 Spark	两个在云计算中工具用来执行 MapReduce 操作的工具。Hadoop 更适合于已经定义和存储的数据集，而 Spark 更适合于你有流式、特别的大或动态生成的数据时
Celery（任务队列使用和管理）	让你能够使用 Python 创建一个任务队列并管理它，允许你自动化没有清晰的开始和结束时间的任务
`logging` 模块	用于你的应用或脚本的内置日志模块，通过它你可以轻松地跟踪错误、调试信息和异常
`smtp` 和 `email` 模块	来自 Python 脚本的内置邮件警报
Twilio	一个有着 Python API 客户端的服务，提供电话和文本信息业务
HypChat	一个 Python API 库，可以使用 HipChat 聊天客户端构建聊天程序
日志服务	使用类似 Sentry 或 Logstash 的服务管理你的日志、错误率和异常
监控服务	使用类似 New Relic 或 Datadog 的服务监控日志、服务正常运行时间、数据库问题和性能（即发现硬件问题）

学习了本书前面章节中的丰富知识，你现在应该已经准备好去花时间构建高质量的工具，并让这些工具为你做枯燥乏味的工作了。你可以丢掉那些老套的电子表格公式，使用 Python 导入数据，运行分析，直接交付报告到邮箱。你可以真正地让 Python 管理刻板的任务（就像机器人助手一样），自己投身于报告中更关键和有挑战的部分。

第 15 章

结论

恭喜！你已经到达本书的最后一章。在刚开始阅读本书的时候，你可能只知道一点 Python 知识，也没有使用编程研究数据的经验。

现在你的经验应该相当不同了。你已经积累了寻找和清洗数据的知识和经验。你已经通过专注于问题和根据给定的数据集进行取舍，磨练了技能。你可以编写简单的正则表达式和复杂的网页抓取器。你可以在云端规模化数据和进程，并通过自动化的方式管理数据。

然而，乐趣不会止于此！作为数据处理者，你在职业生涯中还有很多要学习和实践的东西。你可以利用在本书中学到的技能和工具，继续扩展知识，进而拓展数据处理领域的界限。我们鼓励你追求卓越，继续针对数据、进程和方法提出难题。

15.1 数据处理者的职责

正如我们在本书和调查中确定的，数据和数据处理者可得出的结论是非常多的。但机会也伴随着责任。

没有数据处理的警察；但是，你在我们的书中学到了一些道德知识。你已经学到要成为一名谨慎的网页抓取者。你学习了拿起电话，询问更多的信息；学习了在展示发现时如何解释和文档化你的进程；还学习了如何针对困难话题提出难题，特别是当数据源有其他动机的时候。

作为一个数据处理者，随着你不断学习、成长，你的道德感也会增强，并会在工作和进程中指引和帮助你。从某种程度上讲，你现在已经是一名研究型记者了。你获得的结论和你问的问题可以并且能够在你的领域中产生影响。知道了这些，你就有了责任的压力。

你的职责包括：

- 使用你的知识、技术和能力，用于公正和有益的事情；
- 为你周围的人贡献知识；
- 回馈帮助你的社区；
- 挑战迄今为止你学到的道德的反面，并且继续开发。

我们鼓励你在数据处理者的职业生涯中迎接这些挑战。你是否喜欢同其他人一起工作，分享知识？成为一名导师！你是否喜欢某个开源包？成为一个代码或文档的贡献者！你是否研究过重要的社会或健康问题？将你的发现贡献给学术团体或社区！你是否经历过来自某个社区或代码的难题？向全世界分享你的故事吧。

15.2　数据处理之上

本书的课程提高了你的技能，但是你还有很多要学习的东西。根据你的技术栈和兴趣，有许多领域值得进一步探索。

15.2.1　成为一名更优秀的数据分析师

本书介绍了统计和数据分析。如果你想要真正地夯实统计和分析技能，需要花更多的时间阅读方法背后的科学知识，同样还要学习一些更专业的 Python 包，这样你在分析数据集时会有更多的能力和灵活性。

为了学习更高级的统计学知识，回归模型和数据分析背后的数学知识是必须学习的。如果你没有上过任何统计学课程，Edx 有一个很棒的来自加州大学伯克利分校的存档课程（https://courses.edx.org/courses/BerkeleyX/Stat_2.1x/1T2014/info）。如果你想通过读书来探索，Allen Downey 的《统计思维：程序员数学之概率统计（第 2 版）》[1] 很好地介绍了统计数学概念并且使用了 Python。Cathy O'Neill 和 Rachel Schutt 的《数据科学实战》[2] 提供了数据科学领域的深入分析。

如果你想学习 `scipy` 技术栈和更多关于 Python 如何帮助你进行高级数学和统计分析的知识，你很幸运。`pandas` 的一个主要贡献者 Wes McKinney 编写了《利用 Python 进行数据分析》，深度讲解了 `pandas`。`pandas` 的文档（http://pandas.pydata.org/pandas-docs/stable/10min.html）也是很好的入门方法。在第 7 章中，你学习了一些关于 `numpy` 的知识。如果你对学习 `numpy` 一些核心的知识感兴趣，查看 SciPy 关于基础的介绍（https://docs.scipy.org/doc/numpy/user/basics.html）。

15.2.2　成为一名更优秀的开发者

如果你想要增强 Python 技能，Luciano Ramalho 的《流畅的 Python》[3] 深入讨论了 Python 中的一些设计模式。我们也强烈建议你浏览世界各地最新的 Python 活动的视频（http://pyvideo.org/），并研究你感兴趣的主题。

注 1：此书已由人民邮电出版社出版。——编者注
注 2：此书已由人民邮电出版社出版。——编者注
注 3：此书已由人民邮电出版社出版。——编者注

如果本书是你的第一本编程入门书，你可能想要参加一个计算机科学的入门课程。如果你想要自学，Coursera 提供了斯坦福大学的一个课程（https://www.coursera.org/course/cs101）。如果你想要一本介绍计算机科学背后的理论知识的在线教科书，我们推荐 *Structure and Interpretation of Computer Programs*（https://mitpress.mit.edu/sicp/full-text/book/book.html），由 Harold Abelson 和 Gerald Jay Sussman 编写（MIT 出版社）。

如果你想通过与其他人一起构建和工作来学习更多的开发原则，我们建议你找一个本地讨论组并参与进去。许多类似的小组会举办本地或远程的极客开发活动，所以你可以同他人一起编写代码，通过实践来学习。

15.2.3 成为一名更优秀的视觉化讲故事者

如果你对本书中视觉化讲故事的部分特别感兴趣，有很多拓展该领域知识的方式。如果你想继续研究我们使用过的库，我们强烈建议你学习 Bokeh 的入门教程（http://bokeh.pydata.org/en/latest/docs/user_guide/tutorials.html），同时利用你的 Jupyter notebooks 做实验。

学习 JavaScript 和 JavaScript 社区中流行的一些可视化库，会帮助你成为一名更好的用视觉讲故事者。Square 提供了一个关于 D3 课程的介绍（https://square.github.io/intro-to-d3/），对流行的 JavaScript 库 D3（https://d3js.org/）做了简短说明。

最后，如果你想从数据分析的角度学习视觉化讲故事背后的一些理论和想法，我们推荐 Edward Tufte 的 *Visual Display of Quantitative Information*（https://www.amazon.com/Visual-Display-Quantitative-Information/dp/0961392142/，Graphics 出版社）。

15.2.4 成为一名更优秀的系统架构师

如果你想学习如何规模化、部署和管理系统，我们几乎没有触及系统表层。

如果你想学习更多的 Unix 知识，萨里大学有一个简短的教程，介绍了一些优秀的概念（http://www.ee.surrey.ac.uk/Teaching/Unix/index.html）。Linux 文档项目同样有一个简短的关于 bash 编程的介绍（http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html）。

我们强烈建议你花时间学习 Ansible（http://docs.ansible.com/ansible/intro_getting_started.html），一个可扩展的、灵活的服务器和系统管理解决方案。如果你对规模化数据解决方案感兴趣，Udacity 提供了一个对 Hadoop 和 MapReduce 课程的介绍（https://cn.udacity.com/course/intro-to-hadoop-and-mapreduce--ud617）。你同样应该查看斯坦福大学关于 Apache Spark 的介绍（http://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf），还有 PySpark 编程指南（https://spark.apache.org/docs/0.9.0/python-programming-guide.html）。

15.3 下一步做什么

下一步你会怎么做？你拥有很多新技能，并且能够质疑自己的假设和发现的数据。你还有一些 Python 的应用知识，而且很多有用的库触手可及。

如果你对特定的领域或数据集没有热情，你会想要发现新的学习领域，继续你作为一名数

据处理者的进程和探索。很多很棒的数据分析师书写了很多激励人心的故事。这里就有一些这样的故事。

- FiveThirtyEight（http://fivethirtyeight.com/），曾是《纽约时报》记者 Nate Silver 的博客，现在是一个拥有众多作家和分析师的站点，研究许多不同的主题。在弗格森大陪审团决定不起诉达伦· 威尔逊后，FiveThirtyEight 发布了一篇文章，显示收入出现了离群现象（https://fivethirtyeight.com/datalab/ferguson-michael-brown-indictment-darren-wilson/）。 在一个争议性的话题中，如果有能力展示数据趋势或倾向，将可以帮助我们去除故事中的一些情绪，揭示数据真正想要诉说的东西。
- 《华盛顿邮报》的一项关于收入差距的研究（https://www.washingtonpost.com/news/wonk/wp/2014/11/04/affirmative-action-for-rich-kids-getting-a-job-at-dads-company/?utm_term=.80495391806a）使用税收和人口普查数据得出结论：在择业和初始薪水方面“老同学关系网”（ol’boy network）依然存在，但是在得到第一份工作后，这些关系网通常会弱化或者消失。
- 我们已经研究了非洲雇用童工的组织带来的一些影响，包括挖掘冲突矿产。国际特赦组织（Amnesty International）和全球见证（Global Witness）（http://www.bbc.com/news/business-32403315）近日的一个报道发现，大多数美国公司没有充分检查供应链，来确保没有使用冲突矿产。

这个世界还有数不清的未讲述的故事。如果你有激情或信念，很可能你的观点和数据处理技术可以帮助人们和社区。如果你没有激情，我们鼓励你通过跟进新闻、文档和网络上的数据分析，不断地学习。

无论你的兴趣是什么，都有无限的可能性去深入学习和掌握本书中介绍的概念。任何最能激发你兴趣的事情，就是未来学习的最好途径。我们希望这本书只是你数据处理者生涯的一个起点。

附录A

编程语言对比

通常情况下，当你使用某种编程语言时，其他人都会问你选择这种语言的原因。为什么不用 X 或 Y 呢？ X 或 Y 的选择有许多，这取决于提问者都知道哪些语言，也取决于他是不是一个狂热的开发者。要试着理解他们问这个问题的原因，也最好思考一下自己的回答：为什么选择 Python ？本附录将对比 Python 和其他有用的编程语言，你可以据此回答上述问题，也可以深入理解我们选择编程语言的理由。

A.1　C、C++、Java与Python

与 C、C++ 和 Java 相比，Python 很容易学习，特别是对没有计算机背景的人来说。许多和你起点相同的人已经可以开发插件和有用的工具，使 Python 在数据科学和数据处理领域更加强大、更加高效。

至于技术性差异，Python 是一种高级语言，而 C 和 C++ 是低级语言。Java 是高级语言，但又有一些低级语言的特性。这是什么意思？高级语言将与计算机架构的交互抽象化，你可以输入代码字（code word，比如 `for` 循环或变量定义），高级语言会将其编译成计算机可以执行的代码，而低级语言会直接处理这些代码字。与高级语言相比，低级计算机语言的运行速度更快，可以更直接地控制系统，从而优化内存管理等。高级语言更容易学习，因为它已经帮你完成了大多数低级别的任务。

对于本书中的练习而言，不需要系统控制，也不需要将运行时间缩短数秒，所以我们不需要用到低级语言。虽然 Java 也是高级语言，但它的学习难度比 Python 更高，你需要更长时间才能上手。

A.2　R或MATLAB与Python

Python 有许多库（补充代码），能够实现与 R 和 MATLAB 相同的功能。这些库的名字是 `pandas`（http://pandas.pydata.org/）和 `numpy`（http://www.numpy.org/）。这些库可以处理与大数据和统计分析相关的任务。如果你想学习更多这方面的内容，你应该查阅 Wes McKinney 的《利用 Python 进行数据分析》一书。如果你在 R 或 MATLAB 方面有较强的专业背景，可以继续用这些工具来处理数据。在这种情况下，Python 是一个很好的补充工具。但是，用同一种语言负责工作流程中的每一个环节，可以让数据处理过程更容易，维护起来也更加方便。同时学习 R（或 MATLAB）和 Python，你可以根据具体项目需求来选择使用哪种语言，这样可以更好地满足项目的需求，也更加方便。

A.3　HTML与Python

解释为什么不用 HTML 处理数据，就如同解释为什么不把水放到油箱里一样，你就是不会这么做。HTML 的用途不在于此。HTML 代表 HyperText Markup Language（超文本标记语言），用来为浏览器中显示的网页提供基本结构。我们在第 3 章中讨论过 XML，与此类似，我们也可以用 Python 解析 HTML，但反过来却不行。

A.4　JavaScript与Python

JavaScript 是一种向网页添加交互与功能的语言，不要与 Java 混为一谈。JavaScript 运行在浏览器中。Python 与浏览器分离，运行在计算机系统上。Python 有大量库，可以充实数据分析的功能。JavaScript 具有与浏览器相关的额外功能。你可以利用 JavaScript 抓取网络数据并创建图表，但无法用于统计汇总。

A.5　Node.js与Python

Node.js 是一个 Web 平台，而 Python 是一种语言。有许多用 Python 编写的框架与 Node.js 类似，像 Flask 和 Django，但 Node.js 是用 JavaScript 语言编写的。Node.js 主要用的是 JavaScript，所以你可以用 JavaScript 作为后端语言。如果你后端用的是 Flask 或 Django，你可能需要学习 JavaScript 来处理前端需求。但本书大部分内容针对的都是后端过程和大型数据处理过程。Python 更方便、更容易学习，还有现成的数据处理库。正因为这一点，我们选择用 Python。

A.6　Ruby和Ruby on Rails与Python

你可能听说过 Ruby on Rails，这是基于 Ruby 语言的一个流行的 Web 框架。Python 也有许多框架：Flask、Django、Bottle、Pyramid 等，Ruby 也经常不用于 Web 框架。我们之所以选择 Python，是因为它快速的数据处理能力，而不是因为它的 Web 框架能力。我们的确会讲到数据展示，但如果你的目标是创建一个网站的话，那么你不应该阅读本书。

附录B

初学者的Python学习资源

本附录列出了许多适合 Python 开发新人的学习小组和学习资源。这个列表并不全面，但它介绍了许多网站、论坛、聊天室和线下小组，在你追求成为优秀 Python 开发者的过程中可以访问并使用这些资源。

B.1　在线资源

- Stack Overflow（http://stackoverflow.com/）是一个非常有用的网站，你可以在上面提问与编程和 Python 相关的问题，查看其他人的问题和回答并给出自己的回答。这个网站支持在问题中发布代码、对回答点赞，还可以在已经问过的大量问题中搜索。如果你在学习过程中遇到困难，Stack Overflow 上的某个答案可能会给你一些提示！
- Python 官网（http://python.org/）是一个好工具，用来深入研究开发过程中可能用到哪些库。如果你对 Python 标准库中的某些原理有疑问，或者不知道都有哪些推荐使用的外部库，可以先从 Python 官网开始查看。
- Read the Docs（https://readthedocs.org/）是一个有用的网站，许多 Python 库都会将文档保存在这里。如果你想寻找某个库的更多使用方法，可以在这个网站上查找。

B.2　线下小组

- PyLadies（http://www.pyladies.com/）是一个女性工程师小组，旨在推动 Python 的多样性。PyLadies 在世界各地都有分会，在 freenode 网站上有一个活跃的 IRC 频道，在 PyLadies 网站上还有大量的研讨会和其他有用工具。大多数分会向所有会员开放，但要看当地分会的线下聚会是否要求特定性别才能参加。

- Boston Python（http://www.meetup.com/bostonpython/）是世界上最大的 Python 线下聚会之一。这个小组由一个知名开发者和教育工作者组成的动态小组管理，帮助组织研讨会、项目之夜，以及各种不同的教育活动。如果你在波士顿地区生活，你可以亲自去感受一下！
- PyData（http://pydata.org/）是一家帮助创建 Python 社区和数据分析社区的机构。它们在世界各地举行线下聚会和大型会议，你在当地可能也会找到一个线下分会（或者你也可以自己创建一个新的分会）。
- Meetup.com（http://www.meetup.com/）是许多技术相关的教育活动的发布网站。我们推荐搜索当地与 Python 和数据相关的线下聚会。注册网站很简单，根据你的兴趣找到的新线下聚会将会向你发送提醒，你会遇见很多人，他们和你一样都对 Python 和数据感兴趣。
- Django 女孩（https://djangogirls.org/）是一个女性工程师小组，旨在推动利用 Python 大型 Web 开发框架 Django 进行 Python 开发。世界各地有许多活跃的分会，提供各种研讨会和培训班。

附录C

学习命令行

有一个强大的开发工具，就是只用命令行来控制计算机的能力。无论你用的是哪种操作系统，懂得直接与计算机交互的方法，可以让你的数据处理任务和编程任务事半功倍。我们并不是说你要成为系统管理员，但擅于通过命令行来控制计算机是很有用的。

作为一名开发人员，最有成就感的事情之一就是能够对遇到的系统问题和代码问题进行调试。通过命令行理解计算机的运行原理，可以让你深入理解这些问题。如果你遇到了系统错误，并使用本书学到的调试技巧，你可能会更了解自己的计算机和用到的操作系统，也更了解如何用命令行更好地交互。在 Python 代码中遇到系统错误时，你就会在调试并解决这些问题时快人一步。

本附录中我们会讲到 bash（在 Mac 和许多 Linux 上使用）的基础知识，也会讲到 Windows 上 cmd 和 PowerShell 程序的基础知识。我们这里只是做一个简要的介绍，但我们建议你继续深入学习。在每一小节我们都给出了进一步阅读的建议。

C.1　bash

如果你用的是基于 bash 的命令行，你在 C.1.1 节学到的内容也同样适用于任意基于 bash 的客户端，这和你当前使用的操作系统无关。太酷了！ bash 是一种功能强大的 shell 语言（或命令行语言）。我们开始学习 bash，首先来看如何在计算机中浏览文件。

C.1.1　跳转命令

在命令行中浏览计算机可以帮你理解用 Python 如何做到这一点。停留在终端或文本编辑器中，可以让你保持专注。

我们首先来看基本命令。打开终端。打开的终端可能会位于 ~ 文件夹，代表你的 home 目录。如果你用的是 Linux，home 目录可能位于 /<home/your_computer_name>。如果你用的是 Mac，home 目录可能位于 /Users/<your_name>。想要查看你所在的文件夹，可以输入：

```
pwd
```

你应该会看到像这样的输出：

```
/Users/katharine
```

或者像这样：

```
/home/katharine
```

`pwd` 代表“打印工作目录”（print working directory）。你让 bash 告诉你当前所在的文件夹（或目录）。如果你是第一次学习用命令行跳转，那么这条命令很有用，特别是你想再次确认自己是否位于正确的文件夹中时。

另一个有用的命令用来查看文件夹中都有哪些文件。想要查看当前工作目录中都有哪些文件，输入：

```
ls
```

你应该会看到像这样的输出：

```
Desktop/
Documents/
Downloads/
my_doc.docx
...
```

具体内容可能会有所不同，颜色也可能不同，这与你的操作系统有关。`ls` 的含义是“列出清单”（list）。在调用 `ls` 时还可以指定额外参数，叫作*标记*（flag）。这些参数会改变输出的内容。试一下这个命令：

```
ls -l
```

输出应该有许多列，最后一列与只输入 `ls` 的输出相同。`-l` 标记给出了目录的详细（长）内容，其中包括所包含的文件和目录的数目，以及每一个文件和目录的权限、创建者的名字、组所有权、大小和最后修改日期。下面给出一个实例：

```
drwxr-xr-x  2 katharine katharine  4096 Aug 20  2014 Desktop
drwxr-xr-x 22 katharine katharine 12288 Jul 20 18:19 Documents
drwxr-xr-x 26 katharine katharine 24576 Sep 16 11:39 Downloads
```

这些细节可以帮你发现与权限有关的问题，还可以查看文件大小和其他信息。你还可以向 `ls` 传入任意目录，它都可以列出里面的内容。试着查看下载文件夹的内容（输入以下代码）：

```
ls -l ~/Downloads
```

你会看到与前面类似的很长的输出，但列出的是 Downloads 文件夹下的所有文件和目录。

现在你学会了如何列出不同文件夹下的文件，下面我们来学习如何改变当前所在文件夹。我们利用 `cd` 命令来“改变目录”（change directory）。试着输入：

```
cd ~/Downloads
```

现在用 `pwd` 查看位于哪个文件夹，用 `ls` 查看文件夹中的文件，你应该会发现正位于 Downloads 文件夹下。如果你想回到主文件夹的话，应该怎么做？我们知道主文件夹是上一级文件夹。你可以用 `..` 跳转到上一级文件夹。试着输入：

```
cd ..
```

现在你回到了主文件夹。在 bash 中，`..` 的意思是“跳转到上一级目录”。你还可以连用两次，跳转到上两级目录，像这样：`cd ../..`。

> 在命令行中跳转目录或选择文件时，你应该可以用 Tab 键自动补全文件名和文件夹名。对于你想选择的文件或文件夹，先输入名字的第一个字母或前两个字母，然后只需按下 Tab 键，你应该会看到不同的匹配选择（帮你完成名字的拼写）。如果没有类似名字的其他文件，命令行会将名字自动补全。这个方法可以节省很多打字的时间！

现在你应该更习惯于用命令行来跳转目录。下面我们将学习如何用命令行移动文件和修改文件。

C.1.2 修改文件

利用 bash 移动文件、复制文件和创建文件都很简单。我们先来看创建新文件。首先，跳转到 home 目录（`cd ~`）。然后输入下列内容：

```
touch test_file.txt
```

然后再继续输入 `ls`。你应该会看到有一个叫作 test_file.txt 的新文件。`touch` 可以用来创建新文件。这个命令会寻找叫这个名字的文件。如果有这个文件的话，会改变最后一次修改的时间戳，但不会修改文件内容；如果文件不存在，会创建这个文件。

> **Atom shell 命令**
>
> 如果你的文本编辑器是 Atom.io（https://atom.io/），你可以利用下面这个命令轻松将这个文件（或任何文件）用 Atom 打开：
>
> ```
> atom test_file.txt
> ```
>
> 如果报错的话，你可能没有安装命令行选项。安装方法是：按 Shift-Cmd-P 键打开命令面板，然后运行 Install Shell Commands 命令。
>
> 想要查看 Atom 所有命令行选项，可以输入 `atom --help`。

对于之前创建的文件，我们试着将其复制到下载文件夹中：

```
cp test_file.txt ~/Downloads
```

这里我们的意思是："将 test_file.txt 复制到 ~/Downloads 文件夹中。"bash 知道 ~/Downloads 是一个文件夹，所以会自动将文件复制到那个文件夹里面。如果我们想要在复制文件的同时改变文件名，可以这么做：

```
cp test_file.txt ~/Downloads/my_test_file.txt
```

这个命令的作用是让 bash 将测试文件复制到下载文件夹中，并将复制后的文件命名为 my_test_file.txt。现在你的下载文件夹中应该有测试文件的两份副本：一个用的是原来的名字，一个用的是新名字。

> 如果你需要多次运行同一个命令，只需按向上键，在命令行历史中查找。如果你想查看最近所有的命令行历史，可以输入 `history`。

有时你并不想复制文件，而是想要移动文件或重命名文件。在 bash 中，我们可以用同一个命令来移动文件和重命名文件：`mv`。我们首先对在主文件夹中创建的文件进行重命名：

```
mv test_file.txt empty_file.txt
```

这里我们告诉 bash："将名为 test_file.txt 的文件移动到名为 empty_file.txt 的文件。"输入 `ls`，你应该会看到已经没有 test_file.txt 了，但现在有一个 empty_file.txt。我们只是利用"移动"来对文件重命名。我们还可以利用 `mv` 在文件夹之间移动文件：

```
mv ~/Downloads/test_file.txt .
```

这里我们的意思是："将下载文件夹中的 test_file.txt 移动到这里。"在 bash 中，`.` 代表你的工作目录（就像 `..` 代表当前目录的"上一级"文件夹一样）。现在输入 `ls`，你应该可以看到主文件夹中又有一个 test_file.txt 文件了。你还可以输入 `ls ~/Downloads`，现在这个文件已经不在下载文件夹中了。

最后，你可以用命令行删除文件。你可以用 `rm`（remove）命令来做到这一点。试着输入：

```
rm test_file.txt
```

现在再输入 `ls`，你会发现 test_file.txt 已经从文件夹中删掉了。

> 与用鼠标删除文件不同，用命令行删除文件是真正的删除。没有"回收站"可以恢复文件，所以使用 `rm` 时一定要小心，对你的计算机和代码一定要定期按时备份。

现在你知道利用 bash 如何移动、重命名、复制和删除文件，下面我们继续学习在命令行中运行文件。

C.1.3 运行文件

利用 bash 运行文件是相当简单的。在第 3 章中你可能已经学过运行 Python 文件，你只需

运行：

```
python my_file.py
```

其中 my_file.py 是一个 Python 文件。

> 对于编程用到的大多数语言来说，只输入语言的名字（`python`、`ruby`、`R`）和文件名（并带有正确的文件路径或文件位置）就可以运行。如果你在用某种语言运行文件时遇到问题，我们建议以语言的名字和“命令行选项”（command-line options）为关键词在网络上进行搜索。

作为一名 Python 开发者，你还会用到其他运行命令。表 C-1 中给出了其中一些命令，你可以先了解一下，在安装和运行外部库时可能会用到。

表C-1：bash中的运行命令

命令	使用案例	更多文档
`sudo`	以 sudo 或（超级）用户的身份运行下面的命令。在修改文件系统的核心部分或安装外部包时通常需要用到 `sudo`	https://en.wikipedia.org/wiki/Sudo
`bash`	运行 bash 文件，或回到 bash shell	http://ss64.com/bash/
`./configure`	运行软件包的配置设定（从源代码安装软件包的第一步）	https://en.wikipedia.org/wiki/GNU_build_system#GNU_Autoconf
`make`	配置完成后运行 `makefile`，编译代码准备安装（从源代码安装软件包的第二步）	http://www.computerhope.com/unix/umake.htm
`make install`	运行 `make` 编译好的代码，将软件包安装到计算机中（从源代码安装软件包的第三步）	http://www.codecoffee.com/tipsforlinux/articles/27.html
`wget`	访问 URL，并下载 URL 里包含的文件（适用于下载软件包或文件）	http://www.gnu.org/software/wget/manual/wget.html
`chown`	改变文件或文件夹的所有权。通常与 `chgrp` 一起使用，用来改变文件的所属群组。如果你需要移动文件让另一个用户可以运行这些文件，那么这个命令是很有用的	http://linux.die.net/man/1/chown
`chmod`	改变文件或文件夹的权限，通常是使文件可执行，或对另一类用户或用户组可见	http://ss64.com/bash/chmod.html

在使用命令行的过程中，你可能会遇到其他各种各样的命令和文档。我们建议你花点时间去学习、使用并提问。bash 是一种不同的语言，需要慢慢去学习它的特点和用法。最后，我们要向你介绍利用 bash 来搜索文件或文件内容。

C.1.4　利用命令行进行搜索

在 bash 中，搜索文件和在文件内搜索都相对简单，也有很多种方法。我们将向你展示几种方法。首先，我们将会用一个命令搜索文件中的文本。我们先用 `wget` 下载一个文件：

```
wget http://corpus.byu.edu/glowbetext/samples/text.zip
```

这应该会下载一个文本语料库，我们可以在里面搜索。将文本解压到一个新文件夹，只需输入：

```
mkdir text_samples
unzip text.zip text_samples/
```

现在在新文件夹 text_samples 中应该有许多文本语料库文件。输入 `cd text_samples` 切换到这个目录中。我们利用一个叫作 `grep` 的工具来搜索这些文件的内容：

```
grep snake *.txt
```

这里的命令是告诉 bash，在这个文件夹所有以 .txt 结尾的文件中搜索字符串 snake。在 7.2.6 节可以学到更多关于通配符的内容，但 * 几乎总可以用作通配符，意思是“任意匹配的字符串”。

运行上面的命令，你应该会看到输出大量匹配的文本。`grep` 会返回所有匹配文件中包含目标字符串的所有行。比如当你要修改某个函数，想在一个很大的仓库中找到包含这个函数的文件时，这一命令特别有用。如果你想打印出前后的几行，还可以向 `grep` 传入额外的参数和选项。

> 想查看任意 bash 命令的选项，只需先输入该命令，然后输入空格和 `--help` 即可。输入 `grep --help`，然后阅读 `grep` 的一些额外选项和特性。

另一个简洁的工具是 `cat`。它只是打印出你指定的文件内容。这个命令很有用，特别是需要将输出内容通过“管道”（pipe）传递到其他地方时。在 bash 中，| 字符可以将你希望对文件或文本执行的一系列操作合并在一起。例如，我们将其中一个文件的内容 `cat` 出来，然后用 `grep` 在输出中搜索：

```
cat w_gh_b.txt | grep network
```

首先返回的是 w_gh_b.txt 文件的所有文本，然后通过“管道”将输出传递给 `grep`，然后在其中搜索单词 network 并在命令行中返回包含该词的文本行。

我们还可以对 bash 历史记录做同样的管道操作。试着输入：

```
history | grep mv
```

这一命令可以帮你找到在学习 bash 的过程中用过的命令并复用，你可能已经不记得这些命令了。

我们来进一步学习搜索，学习如何查找文件。首先，我们将用到一个叫作 `find` 的命令，用来查找匹配的文件名，还可以遍历所有子目录并查找匹配的文件。我们在当前文件夹及其子文件夹中找出所有文本文件：

```
find . -name "*.txt" -type f
```

这一命令的作用是，（在这个文件夹及其所有子文件夹中）寻找相应的文件，文件名以 .txt

结尾，文件类型为 f（标准文件，而不是目录，目录的类型符号为 d）。你应该会看到输出许多匹配的文件名。下面，我们将这些文件通过管道传递给 grep 命令：

```
find . -name "*.txt" -type f | xargs grep neat
```

这里我们让 bash 做的是，“找到与上面相同的文本文件，然后在这些文件中搜索单词 neat”。我们用到了 xargs 命令（https://en.wikipedia.org/wiki/Xargs），这样才能将 find 的输出通过管道正确地传递给 grep。不是所有的管道操作都需要用到 xargs，但当 find 命令的输出结果不完全相同时，这一命令十分有用。

你已经学会搜索和查找的一些实用技巧，特别是你要处理的代码和项目越来越庞大、越来越复杂时，这些技巧尤其有用。关于这一话题，我们还为你提供了更多资源和阅读材料。

C.1.5　更多资源

在互联网上有许多优秀的 bash 学习资源（http://wiki.bash-hackers.org/scripting/tutoriallist）。Linux 文档计划（http://www.tldp.org/LDP/Bash-Beginners-Guide/html/）中也有许多适合初学者的优秀教程，里面会讲到一些高级的 bash 编程知识。O’Reilly 出版了一本很棒的 *bash Cookbook*（http://shop.oreilly.com/product/9780596526788.do），你也可以用这本书开始学习。

C.2　Windows cmd/PowerShell

Windows 命令行（现在再加上 PowerShell，https://en.wikipedia.org/wiki/Windows_PowerShell），或者叫 cmd，是一个功能强大的基于 DOS 的程序。在不同版本的 Windows 和 Windows 服务器中，语法是完全相同的。无论你学的是 Python 还是其他语言，学会这些语法都可以让你成为一个更加优秀的程序员。

C.2.1　跳转命令

用 cmd 浏览文件非常简单。首先打开 cmd 程序查看当前目录。输入：

```
echo %cd%
```

这个命令的作用是，告诉 cmd 你想要返回（或输出）%cd%，即当前目录（current directory）。你应该会看到像这样的输出：

```
C:\Users\Katharine>
```

想列出当前目录的所有文件，输入以下代码：

```
dir
```

你应该会看到类似这样的输出：

```
13.03.2015  16:07    <DIR>        .ipython
11.09.2015  19:05    <DIR>        Contacts
11.09.2015  19:05    <DIR>        Desktop
11.09.2015  19:05    <DIR>        Documents
```

```
11.09.2015  19:05    <DIR>          Downloads
11.09.2015  19:05    <DIR>          Favorites
10.02.2014  15:15    <DIR>          Intel
11.09.2015  19:05    <DIR>          Links
11.09.2015  19:05    <DIR>          Music
11.09.2015  19:05    <DIR>          Pictures
13.03.2015  16:26    <DIR>          pip
11.09.2015  19:05    <DIR>          Saved Games
```

`dir` 也有许多可用的选项（http://ss64.com/nt/dir.html），可以排序、分组或显示更多信息。我们来看一下 Desktop 文件夹。

```
dir Desktop /Q
```

我们让 cmd 显示 Desktop 目录中的所有文件，以及这些文件的所有者。你应该会看到文件名的前半部分是每个文件的所有者（比如 `MY-LAPTOP\Katharine\backupPDF`）。在查看文件夹和文件时这一命令非常有用。还有许多好用的选项，可以显示子文件夹，或者按最后一次修改的时间戳排序。

输入下列代码跳转到 Desktop 文件夹：

```
chdir Desktop
```

现在输入 `echo %cd%` 查看当前目录，你应该会发现与之前的不同。想跳转到上一级文件夹，只需输入 `..`。例如，如果我们想跳转到当前目录的上一级文件夹，可以输入：

```
chdir ..
```

你还可以把多个“上一级文件夹”符号连起来（`chdir ..\..` 可以跳转到上两级文件夹，以此类推）。根据你的文件结构不同，如果当前文件夹没有上一级目录的话，你可能会得到一个错误（比如你位于文件系统的根目录下）。

想返回 home 目录，只需输入：

```
chdir %HOMEPATH%
```

你应该回到了我们开始所在的第一个文件夹。现在我们可以用 cmd 在目录间跳转，下面我们继续学习创建、复制和修改文件。

C.2.2 修改文件

首先，我们来创建一个可用于修改的新文件：

```
echo "my awesome file" > my_new_file.txt
```

用 `dir` 查看文件夹中的文件，现在你应该可以看到 my_new_file.txt。在文本编辑器中打开这个文件，你可以看到文件里面写的是“my awesome file”。如果你用的是 Atom，你可以在 cmd 中直接打开 Atom（参见 C.1.2 节）。

有了这个文件，我们试着将其复制到一个新文件夹中：

```
copy my_new_file.txt Documents
```

现在用下列命令列出 Documents 文件夹中所有文件：

```
dir Documents
```

我们应该会看到，已经将 my_new_file.txt 成功复制到这里。

> 你可以用 Tab 键自动补全文件名和路径，简化输入。试着输入 `copy my`，然后按 Tab 键。cmd 应该能够猜出你想输入的是 my_new_file.txt 文件，自动补全文件名。

我们还想移动文件或重命名文件。想用 cmd 移动文件，我们可以用 `move` 命令。试着输入：

```
move Documents\my_new_file.txt Documents\my_newer_file.txt
```

如果现在列出 Documents 目录下的文件，你应该会发现没有 my_new_file.txt 了，只有一个 my_newer_file.txt。`move` 可用于重命名文件（正如上面我们所做的那样）或者移动文件或文件夹。

最后，你可能想要删除不需要的文件。想用 cmd 做到这一点，你可以用 `del` 命令。试着输入：

```
del my_new_file.txt
```

现在查看当前文件夹下的文件，你应该看不到 my_new_file.txt 了。注意，这个命令会完全删除文件。只有在你完全不需要这个文件时才能用这个命令。定期做硬盘备份是很有用的，以防出现任何问题。

我们已经学过了修改文件，下面我们来学习如何在 cmd 中运行文件。

C.2.3　运行文件

想要在 Windows cmd 中运行文件，通常需要输入语言的名字，然后输入文件路径。例如，想运行一个 Python 文件，你需要输入：

```
python my_file.py
```

只要 my_file.py 位于同一文件夹下，就会运行这个文件。只需在 cmd 中输入 .exe 文件的全名和路径，然后按 Enter 键，就可以运行这个文件。

> 与安装 Python 一样，你需要检查安装好的可执行文件的安装包和文件路径是否包含在 `Path` 变量中（详情可查阅 1.2.2 节）。这个变量为 cmd 保存许多可执行字符串。

想了解更多强大的命令行运行命令，我们推荐学习 Windows PowerShell——一种强大的脚本语言，用来编写脚本并用简单的命令行来运行这些脚本。计算机世界（Computerworld）有一篇很好的 PowerShell 入门文章（http://www.computerworld.com/article/2512066/microsoft-windows-introduction-to-windows-powershell.html）。

想要在命令行中运行安装好的程序，你可以使用 `start` 命令。试着输入：

```
start "" "http://google.com"
```

这个命令应该会打开你的默认浏览器，然后跳转到 Google 首页。更多内容可查阅 `start` 命令的文档（https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/start.mspx?mfr=true）。

我们已经学习了用命令行如何运行文件，下面研究如何搜索和查找计算机中的文件和文件夹。

C.2.4 利用命令行进行搜索

首先下载一个可用的语料库。如果你用的是 Windows Vista 或更新的版本，你应该能够运行 PowerShell 命令。试着输入下列命令来加载 PowerShell：

```
powershell
```

你应该会看到类似这样的新提示符：

```
Windows PowerShell
...
PS C:\Users\Katharine>
```

现在我们已经进入了 PowerShell，我们来下载一个用于查找的文件（注意，这两行命令应该在同一行中输入，被分成两行是为了适应页面宽度）：

```
Invoke-WebRequest -OutFile C:\Downloads\text.zip
 http://corpus.byu.edu/glowbetext/samples/text.zip
```

如果你安装的 PowerShell 不是 3.0 版或更高版本，上面这个命令会报错。如果有报错的话，尝试输入下面这个命令，可适用于更早版本的 PowerShell：

```
(new-object System.Net.WebClient).DownloadFile(
'http://corpus.byu.edu/glowbetext/samples/text.zip','C:\Downloads\text.zip')
```

这些命令是用 PowerShell 将一个单词语料库下载到计算机中。我们创建一个新目录，可以将解压后的文件放在里面：

```
mkdir Downloads\text_examples
```

现在我们要向 PowerShell 中添加一个新函数，用来提取压缩的文件。输入下列命令：

```
Add-Type -AssemblyName System.IO.Compression.FileSystem
function Unzip
{
    param([string]$zipfile, [string]$outpath)

        [System.IO.Compression.ZipFile]::ExtractToDirectory($zipfile, $outpath)
}
```

定义好了这个函数，我们可以用它来解压文件。试着将下载内容解压到新文件夹中：

```
Unzip Downloads\text.zip Downloads\text_examples
```

输入 exit 就可以退出 PowerShell。提示符应该会变回 cmd 的标准提示符。输入 dir Downloads\text_examples，你应该会看到许多下载好的语料库文本文件。我们用 findstr 在这些文件中查找：

```
findstr "neat" Downloads\text_examples\*.txt
```

你应该会看到控制台中滚动输出了许多文本。它们是包含单词 neat 的文本文件中的匹配行。

有时你想搜索特定的文件名，而不是文件中的字符串。你可以用 dir 命令加一个过滤器（filter）来做到这一点：

```
dir -r -filter "*.txt"
```

这应该会找出主文件夹下所有子文件夹中的 .txt 文件。如果你需要继续在这些文件中搜索，可以使用管道操作。| 字符可以将第一个命令的输出通过管道传递给下一个命令。比如说，我们可以用管道找到包含特定函数名的所有 Python 文件，或找到包含特定国家名称的所有 CSV 文件。我们试一下，将 findstr 的输出通过管道传递给 find 命令：

```
findstr /s "snake" *.txt | find /i "snake" /c
```

这段代码的作用是，首先找出包含单词 snake 的文本文件，然后利用 find 计算单词 snake 在这些文件中出现的次数。可以看出，学习更多 cmd 命令及其用法将有助于大大简化数据处理人员和开发人员的很多任务，如搜索文件、运行代码、管理工作等。关于这一话题，本附录介绍了一些内容，你可以在此基础上深入学习。

C.2.5　更多资源

想要学习如何将 cmd 用于日常编程和数据处理工作，有许多可以查找 cmd 命令的优秀在线资源（http://ss64.com/nt/）。

如果你想学习更多 PowerShell 的知识，以及如何用 PowerShell 创建强大的脚本并用于 Windows 服务器和计算机，可以阅读一些教程，如微软的 Getting Started with PowerShell 3.0 教程（https://mva.microsoft.com/en-us/training-courses/getting-started-with-powershell-30-jump-start-8276?l=r54IrOWy_2304984382）。你还可以参考 O'Reilly 出版的 *Windows PowerShell Cookbook*（http://shop.oreilly.com/product/0636920024132.do）开始编写你的第一个脚本。

附录D

高级Python设置

在本书的前文中，我们创建了系统 Python。为什么呢？因为它使用起来快速简单。在开始使用更加复杂的库和工具时，你可能会需要一个更加高级的设置。机器上高级的 Python 设置在你组织项目时是很有帮助的。在你需要同时运行 Python 2.7 和 Python 3+ 时，一个高级设置也很有帮助。

> 在本附录中，我们会带你用专家模式设置 Python 环境。因为这里涉及很多依赖，所以指南中的部分内容与你的经验不符是完全有可能的。为了解决这些问题，我们建议你去互联网寻找或询问如何继续。

我们会首先安装一些核心工具，之后安装 Python（2.7，但是你也可以安装 3+）。最后，我们会安装和设置一些虚拟环境，独立出项目，这样你可以对每一个项目使用不同版本的 Python 库。

本附录中的指南覆盖 Mac、Windows 和 Linux。随着你阅读每一个步骤，仔细跟随你的特定操作系统的指令。

D.1　第1步：安装GCC

GCC（GNU Compiler Collection，GNU 编译器集合）的目的是将用 Python 编写的代码转换为机器可以理解的东西——字节码。

在 Mac 上，GCC 包含在 Xcode（https://developer.apple.com/xcode/）和命令行工具（https://developer.apple.com/downloads/）中。你需要下载其中的任意一个。但是对于这两个方式，你都会需要一个 Apple ID（http://bit.ly/create_appleid）用于下载。同样，Xcode 需要花费一些时间从互联网上下载（我下载了 20 分钟），所以可以计划休息一下。 如果你关心时间

或内存的使用，选择命令行工具取代 Xcode。安装 Xcode 或命令行工具不会花费太长的时间。在继续安装 Homebrew 之前，确保 Xcode 或命令行工具已经安装。

如果你正在使用 Windows，Jeff Preshing 写了一个关于安装 GCC 的很有帮助的教程（http://preshing.com/20141108/how-to-install-the-latest-gcc-on-windows/）。如果你正在使用 Linux，GCC 安装在绝大多数基于 Debian 的操作系统上，或者你可以通过运行 `sudo apt-get install build-essential` 安装它。

D.2　第2步：（只在Mac上）安装Homebrew

Homebrew 管理 Mac 系统上的包，这意味着你可以输入命令，Homebrew 会在安装中帮助你。

> 安装 Homebrew 之前，确保已经安装 Xcode 或命令行工具。否则，你会在安装 Homebrew 时遇到错误。

为了安装 Homebrew，打开控制台，输入这行代码（跟随任何出现的提示，包括请求允许安装 Homebrew 的权限）：

```
$ ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

注意命令的输出。Homebrew 建议运行 `brew doctor` 检查并对任何安装中的问题给出警告。根据你系统状态的不同，你可能有很多需要追查的问题。如果你没有收到警告，那么继续下一个步骤。

D.3　第3步：（Mac系统）告诉系统去哪里寻找Homebrew

为了使用 Homebrew，你需要告诉系统它位于哪里。为此，添加 Homebrew 到 .bashrc 文件，或者你正在使用的其他 shell（即，如果你有一个自定义的 shell，需要在这里添加它）。.bashrc 文件可能在系统中还不存在；如果它存在的话，会隐藏在 home 目录里。

在你输入 `ls` 命令时，所有文件名以一个“.”开始的文件是不会显示的，除非你明确地请求显示所有文件。这一做法的目的有两面性。首先，如果文件是不可见的，你就更不可能错误地删除或编辑它们。其次，这些文件类型并不经常使用，所以隐藏它们使得系统外观更清晰。

让我们看一下，在通过给 `ls` 命令添加额外的命令，展示所有的文件后，我们目录的样子。确保你正位于 home 目录中，然后输入下面的命令：

```
$ ls -ag
```

你的输出会看起来类似于：

```
total 56
drwxr-xr-x+ 17  staff   578  Jun 22 00:08 .
drwxr-xr-x   5  admin   170  May 29 09:49 ..
-rw-------   1  staff     3  May 29 09:49 .CFUserTextEncoding
-rw-r--r--@  1  staff 12292  May 29 09:44 .DS_Store
drwx------   8  staff   272  Jun 10 00:45 .Trash
-rw-------   1  staff   389  Jun 22 00:07 .bash_history
drwx------   4  staff   136  Jun 10 00:35 Applications
drwx------+  5  staff   170  Jun 22 00:08 Desktop
drwx------+  3  staff   102  May 29 09:49 Documents
drwx------+ 10  staff   340  Jun 11 23:47 Downloads
drwx------@ 43  staff  1462  Jun 10 00:29 Library
drwx------+  3  staff   102  May 29 09:49 Movies
drwx------+  3  staff   102  May 29 09:49 Music
drwx------+  3  staff   102  May 29 09:49 Pictures
drwxr-xr-x+  5  staff   170  May 29 09:49 Public
```

我们没有 .bashrc 文件，所以需要创建一个。

如果你有 .bashrc 文件，需要备份一个，以防发生任何问题。通过命令行创建一个 .bashrc 的副本非常简单。直接运行下面的命令来复制 .bashrc 到一个新的文件，.bashrc_bkup:

```
$ cp .bashrc .bashrc_bkup
```

为了创建一个 .bashrc 文件，首先需要确认我们拥有一个 .bash_profile 文件。如果我们添加了一个 .bashrc 文件，而没有 .bash_profile，计算机不会知道拿这个文件做什么。

在开始之前，先检查一下我们是否有 .bash_profile 文件。如果有，它会出现在 `ls -ag` 命令产生的目录列表中。如果没有的话，那么你需要创建它。

如果你有一个 .bash_profile 文件，需要将它备份，这样如果有任何问题，可以恢复至原始的设置。运行下面的命令来复制 .bash_profile 到一个新的文件，名为 .bash_bkup：

```
$ cp ~/.bash_profile ~/.bash_profile_bkup
```

之后运行这个命令来复制它到桌面，同时重新命名它：

```
$ cp ~/.bash_profile ~/Desktop/bash_profile
```

如果你正在使用一个已存在的 .bash_profile 文件，启动编辑器，打开你复制到桌面的那个版本的文件。添加下面的代码到文件末尾。这段代码的意思是“如果这里存在一个 .bashrc 文件，那么使用它”。

```
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

如果还没有 .bash_profile 文件，你需要在编辑器中用这些内容创建一个新文件。保存文件

到桌面，命名为 bash_profile，开头没有句号。

> 确保你检查了 home 目录下不存在 .bash_profile 和 .bashrc 文件。如果它们存在的话，确保跟随指引在继续之前创建原始文件的备份。如果没有做这些的话，当执行下面的命令时，你可能重写原始文件，这可能引发问题。

现在回到控制台，运行下面的命令来重命名文件，并且从桌面移动到 home 目录：

```
$ mv ~/Desktop/bash_profile .bash_profile
```

现在，如果运行 `ls -al ~/`，你会在 home 目录中看到一个 .bash_profile 文件。如果执行 `more .bash_profile`，你会看到代码调用了 .bashrc。

现在有了一个 .bashrc_profile 文件，引用 .bashrc 文件，让我们编辑 .bashrc 文件。首先在文本编辑器中打开当前的 .bashrc 文件或一个新文件。添加下面的代码到 .bashrc 文件的末尾。这会在设置中添加 Homebrew 的位置到你的 `$PATH` 变量。新的 path 会比老的 `$PATH` 优先级更高：

```
export PATH=/usr/local/bin:/usr/local/sbin:$PATH
```

现在，保存这个文件到桌面，命名为 bashrc，没有句号。

为你的代码编辑器使用命令行快捷键

我们在 .bashrc 文件中更新设置时，创建一个快捷方式来从命令行工具启动代码编辑器。这不是必需的，但是会使浏览文件目录和在代码编辑器中打开文件更简单。使用你的 GUI 来浏览文件不会像这样有效。

如果你正在使用 Atom，在安装了 Atom 后，你已经有了可用的快捷方式和 shell 命令（https://discuss.atom.io/t/open-file-project-from-terminal-command-line/1305）。Sublime 同样有在 OS X 上可用的命令（https://www.sublimetext.com/docs/2/osx_command_line.html）。

如果正在使用其他的代码编辑器，你可以尝试输入程序名称来看它是否启动，或者输入程序名和 `--help` 来看它是否具有命令行帮助。我们同样建议搜索“< 你的程序名称 > 命令行工具”，来看是否有任何有帮助的结果。

回到控制台，运行下面的命令来重命名文件，并且将它们从桌面移动到你的 home 目录：

```
$ mv ~/Desktop/bashrc .bashrc
```

这时，如果你运行 `ls -al ~/`，你会看到 home 目录中有 .bashrc 文件和一个 .bash_profile 文件。让我们通过在控制台中打开一个新的窗口来检验 `$PATH` 变量。为了检验这个变量，运行下面的命令：

```
$ echo $PATH
```

你会得到像下面这样的输出：

```
/usr/local/bin:/usr/local/sbin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
```

无论输出是什么，你会看到变量的信息（`/usr/local/bin:/usr/local/sbin`）添加到了我们的 .bashrc 文件中，突出了返回值。

如果你没有在变量中看到新的值，确保你打开了一个新的窗口。设置的改变不会加载到当前的终端窗口，除非你明确地应用文件到当前的终端［查看 bash source（http://ss64.com/bash/source.html）命令获取更多的信息］。

D.4　第4步：安装Python 2.7

为了在 Mac 上安装 Python 2.7，运行下面的命令：

```
$ brew install python
```

如果你想要跟进 Python 3+，你可以安装它。在 Mac 上安装 Python 3+，运行：

```
$ brew install python3
```

对于 Windows，你需要跟随第 1 章中的指引，正确地通过 Windows 安装程序安装。对于 Linux，很可能你已经安装了 Python。在 Linux 中，通过安装一些 Python 开发者包（http://stackoverflow.com/questions/6230444/how-to-install-python-developer-package）来安装一些额外的工具是好的想法。

在完成所有的操作后，测试它是否工作正常。

在终端启动你的 Python 解释器：

```
$ python
```

然后，运行下面的代码：

```
import sys
import pprint
pprint.pprint(sys.path)
```

Mac 输出看起来类似这样：

```
>>> pprint.pprint(sys.path)
['',
 '/usr/local/lib/python2.7/site-packages/setuptools-4.0.1-py2.7.egg',
 '/usr/local/lib/python2.7/site-packages/pip-1.5.6-py2.7.egg',
 '/usr/local/Cellar/python/2.7.7_1/Frameworks/Python.framework/Versions/
 2.7/lib/python27.zip',
 '/usr/local/Cellar/python/2.7.7_1/Frameworks/Python.framework/Versions/
 2.7/lib/python2.7',
 '/Library/Python/2.7/site-packages',
 '/usr/local/lib/python2.7/site-packages']
```

如果正在使用 Mac，你收到的输出应该有很多以 /usr/local/Cellar/ 开头的文件路径。如果没有看见它，你可能没有在控制台窗口重新加载设置。关闭窗口，再打开一个新的窗口，重新尝试一次。如果这没有解决过程中出现的任何问题，返回到设置的最开始，重新执行步骤。

调试安装错误是一个学习与积累经验的过程。如果你遇到了在本节中没有描述的错误，在

浏览器中打开你最喜欢的搜索引擎，搜索这个错误。你可能不是第一个经历该问题的人。

如果你成功完成了本节内容，请继续下一步。

D.5 第5步：安装virtualenv（Windows、Mac、Linux）

我们已经设置了第二个 Python 实例，但是现在我们想要设置一种创建独立的 Python 环境的方式。通过隔离不同的项目和依赖，virtualenv 可以帮助我们达到这个目的。如果有多个项目，我们可以确保每个需求间没有冲突。

为了安装 virtualenv，我们需要 Setuptools（http://pythonhosted.org//setuptools/）。安装完 Python，Setuptools 也就跟着安装了。Setuptools 的一部分是名为 pip 的命令行工具，我们使用它来安装 Python 包。

为了安装 virtualenv（http://virtualenv.readthedocs.org/en/latest/），在命令行中运行下面的命令：

```
$ pip install virtualenv
```

在运行这个命令后，输出的一部分应该为：Successfully installed virtualenv。如果你得到这个信息，那么所有事情运行良好。如果没有，那么你有其他需要解决的问题，可以在网络上搜索，寻求帮助。

D.6 第6步：创建一个新目录

在继续之前，让我们创建一个目录，来维护项目相关的内容。根据个人喜好确定位置。为便于访问和备份，大多数人在用户 home 目录中创建文件夹。你可以将目录放在任何你喜欢的既有用又好记的地方。在 Mac 系统上，要在 home 目录中创建一个 Projects 文件夹，可在终端运行下面的命令：

```
$ mkdir ~/Projects/
```

或者对于 Windows：

```
> mkdir C:\Users\_your_name_\Projects
```

之后在该文件夹中创建一个文件夹，来保存我们编写的数据处理代码。在 Mac 上，你可以通过运行下面的命令达到这个目的：

```
$ mkdir ~/Projects/data_wrangling
$ mkdir ~/Projects/data_wrangling/code
```

对于 Windows，使用下面的代码：

```
> mkdir C:\Users\_your_name_\Projects\data_wrangling
> mkdir C:\Users\_your_name_\Projects\data_wrangling\code
```

最后，在 home 目录中添加一个隐藏文件夹，在 `virtualenv` 环境中使用。在 Mac 上使用下面的命令：

```
$ mkdir ~/.envs
```

对于 Windows 运行下面的命令 :

```
> mkdir C:\Users\_your_name_\Envs
```

如果你想要在 Windows 中隐藏文件夹，可以通过在命令行中编辑属性做到：

```
> attrib +s +h C:\Users\_your_name_\Envs
```

要取消隐藏，可以直接去除属性：

```
> attrib -s -h C:\Users\_your_name_\Envs
```

这时，如果我们查看 Projects 文件夹的内容，应该会看到两个空的子文件夹，名为 code 和 envs。

D.7　第7步：安装virtualenvwrapper

`virtualenv` 是一个很棒的工具，但是 `virtualenvwrapper`（http://virtualenvwrapper.readthedocs.org/en/latest/）让 `virtualenv` 更易于访问和使用。然而它有很多特性（https://virtualenvwrapper.readthedocs.io/en/latest/#features）没有在这个附录中提到，其中最强大的特性也是最简单的特性之一。

它需要一个类似下面的命令：

```
source $PATH_TO_ENVS/example/bin/activate
```

将其变成这样：

```
workon example
```

D.7.1　安装virtualenvwrapper（Mac和Linux）

在 Mac 和 Linux 上安装 `virtualenvwrapper`，运行下面的命令：

```
$ pip install virtualenvwrapper
```

检查输出的倒数第二行，确保所有的安装正确。对我来说，这行代码的意思是成功地安装了 virtualenvwrapper virtualenv-clone stevedore。

更新你的.bashrc

你同样需要添加一些设置到 .bashrc 文件中。我们会复制文件，编辑它，然后将它放回原来的位置。

首先，创建一个 .bashrc 文件的备份。如果已经有了一份，你可以跳过这个步骤。如果你从一个新文件开始，你会创建 .bashrc 文件的第一个备份。为了完成这件事，输入下面的命令：

```
$ cp ~/.bashrc ~/.bashrc_bkup
```

我在 GitHub 上存储了设置文件（https://github.com/jackiekazil/dotfiles），这样我总是有一个可用的备份。在我犯了错误或者计算机死机的时候，我总是可以恢复它。当你不断调整文件，产生 20 个备份时，确保 home 文件夹不会产生混乱。你很少会编辑 .bashrc 文件，但是在编辑这类文件之前，要备份它们。

使用编辑器打开 .bashrc 文件，在文件末尾添加下面三行代码。如果你没有为 Projects 文件夹使用相同的位置，你需要相应地调整文件的路径：

```
export WORKON_HOME=$HOME/.envs                          ❶
export PROJECT_HOME=$HOME/Projects/                     ❷
source /usr/local/bin/virtualenvwrapper.sh              ❸
```

❶ 定义 `WORKON_HOME` 变量。这是存储 Python 环境变量的地方。这应该与你之前创建的环境文件夹匹配。

❷ 定义 `PROJECT_HOME` 变量。这是你存储代码的地方。这应该与你之前创建的 Projects（或 linux 项目）文件夹匹配。

❸ 初始化 `virtualenvwrapper`，它让 `virtualenv` 更容易使用。

完成后，保存文件并打开一个新的终端窗口来加载新的设置。现在你就有了一个易于使用的带有虚拟环境的命令集。

D.7.2　安装`virtualenvwrapper-win`（Windows）

对于 Windows 系统，有一些额外的可选步骤会让你更轻松。首先，你需要安装 Windows 版本的 `virtualenvwrapper`（https://pypi.python.org/pypi/virtualenvwrapper-win）。你同样可以通过运行下面的命令完成这件事：

```
>pip install virtualenvwrapper-win
```

你同样需要添加 `WORKON_HOME` 环境变量。默认情况下，`virtualenvwrapper` 会希望你在 User 文件夹下有一个名为 Env 的文件夹。如果你想要为虚拟环境设置自己的文件夹，那就设置吧，然后添加 `WORKON_HOME` 变量，设置为合适的文件路径。如果你在之前没有设置过环境变量，需要一个快速的指南，在 Stack Overflow 上有一个很棒的教程（https://superuser.com/questions/284342/what-are-path-and-other-environment-variables-and-how-can-i-set-or-use-them）。

为了在 Windows 上与多个版本的 Python 一起工作，安装 `pywin`（https://github.com/davidmarble/pywin）也是一个很好的想法；这便于你在不同版本的 Python 间切换。

D.7.3　测试你的虚拟环境（Windows、Mac、Linux）

在圆满完成这一节之前，让我们运行一些测试程序来确保一切正常工作。在一个新的控制台窗口中，创建一个新的虚拟环境，名为 test:

```
mkvirtualenv test
```

你的输出应该看起来类似这样：

```
New python executable in test/bin/python2.7
Not overwriting existing python script test/bin/python (you must use
  test/bin/python2.7)
Installing setuptools, pip...done.
```

如果你想要使用 Python 3+ 创建环境，而不是 Python 2.7，那么你要定义 `python` 变量，使其指向 Python 3。首先，定义 Python 3 实例的位置：

```
which python3
```

你的输出应该看起来类似这样：

```
/usr/local/bin/python3
```

现在，在你的 `mkvirtualenv` 命令中使用它来定义 Python 3+ 环境：

```
mkvirtualenv test --python=/usr/local/bin/python3
```

你应该会看到“(test)”添加到了控制台提示的开端。这意味着环境现在已经激活。

如果你的输出为 `-bash: mkvirtualenv: command not found`，那么你的控制台没有识别出 `virtualenvwrapper`。首先，检查并确认在运行这段代码前打开了新的控制台或 cmd 窗口，这确保新的设置被应用。如果这不是问题，那么浏览整个过程并确认跟随了所有的步骤。

如果你成功地创建了虚拟环境，那么你完成了设置！

让我们禁用虚拟环境并摧毁它，因为这只是一个测试。运行下面的命令来移除 test 环境：

```
deactivate
rmvirtualenv test
```

这时，你已经在机器上设置了第二个 Python 实例。你还有了一个可以创建独立的 Python 实例的环境，可以在不同的项目间做保护。现在我们会运行一些练习，使你熟悉崭新的 Python 环境。

D.8　学习我们的新环境（Windows、Mac、Linux）

这里展示的例子是面向 Mac 的，但是过程与 Windows 和 Linux 上是相同的。在这一节中，我们会学习一些关于如何使用设置并确保所有的组件一起工作的知识。

让我们通过创建一个名为 testprojects 的新环境开始。我们会在任何需要一个快速的环境来练习测试或做其他事情时，激活并使用它。运行下面的命令来创建它：

```
$ mkvirtualenv testprojects
```

创建环境后，你会看到控制台提示的前面添加了环境名称。对我来说，像下面这样：

```
(testprojects)Jacquelines-MacBook-Pro:~ jacquelinekazil$
```

让我们安装一个 Python 库到环境中。我们会安装的第一个库称为 `ipython`。在你的激活环境中，运行下面的命令：

```
(testprojects) $ pip install ipython
```

如果这个命令是成功的，那么你输出的最后几行看起来像下面这样：

```
Installing collected packages: ipython, gnureadline
Successfully installed ipython gnureadline
Cleaning up...
```

现在，如果你输入 `pip freeze` 到控制台，会看到当前环境的库，同时还有每个安装的版本号。输出会类似下面这样：

```
gnureadline==6.3.3
ipython==2.1.0
wsgiref==0.1.2
```

这些输出告诉我们，在 testprojects 环境中，我们安装了 3 个库：`gnureadline`、`ipython` 和 `wsgiref`。`ipython` 是我们刚刚安装的库。`gnureadline` 是在安装 `ipython` 时安装的库，因为这是一个依赖库。（这让你避免了需要直接安装依赖的库。很好，不是吗？）第三个库是 `wsgiref`。它默认存在，但是不是必需的。

所以我们已经安装了一个叫 `ipython` 的库，但是我们可以用它做些什么呢？ IPython 是一个易于使用的 Python 默认解释器的替代（你可以在附录 F 中读到更多关于 IPython 的信息）。为了启动 IPython，直接输入 `ipython`。

你会看到类似下面的输入提示：

```
IPython 3.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

对其进行测试，输入下面命令：

```
In [1]: import sys

In [2]: import pprint

In [3]: pprint.pprint(sys.path)
```

你应该会看到与我们确认环境工作时相同的输出。`sys` 和 `pprint` 是标准库模块，打包在 Python 之中。

有两种方法退出 IPython。你可以按下 Ctrl+D，在提示时输入 `y`（yes），或者只是输入 `quit()`。它的工作方式同默认的 Python shell 一样。

一旦你退出了，你会回到命令行。现在我们有了一个叫 testprojects 的环境，有 3 个库已经安装。但是如果我们致力于其他项目，想要有另一个环境呢？首先，输入下面的命令禁用当前的环境：

```
$ deactivate
```

之后创建一个新的叫作 sandbox 的环境：

```
$ mkvirtualenv sandbox
```

完成之后，你会进入新环境。如果输入 `pip freeze`，你会看到这个环境还没有安装 IPython。因为这是一个全新的环境，与 testprojects 环境完全分离。如果我们在这个环境中安装 IPython，它会在计算机上安装第二个实例。这保证我们在一个环境中所做的任何事不会影响其他的环境。

为什么它很重要？随着新项目的进行，你会想安装不同的库以及不同版本的库。我们建议为本书创建一个虚拟环境，但是如果你启动了一个新的项目，你会想要创建一个新的虚拟环境。正如你所见，随着项目的切换，在不同的环境中切换是很容易的。

有时候，你可能会碰到所有依赖都存储在一个名为 requirements.txt 文件中的仓库。库作者使用虚拟环境和 `pip freeze` 来保存列表，这样用户可以安装库和依赖。从 requirements 文件安装包，你需要运行 `pip install -r requirements.txt`。

我们知道如何创建和禁用环境，但是我们不知道如何激活一个已经存在的环境。为了激活我们名为 sandbox 的样例环境，输入下面的命令（如果你已经在其中了，需要先 `deactivate` 它，来看有什么区别）：

```
$ workon sandbox
```

最后，你如何摧毁一个环境呢？首先，确保你不在想要删除的环境中。如果你键入了 `workon sandbox`，那么你应该在 sandbox 环境中了。为了删除它，首先你需要禁用它，然后移除它：

```
$ deactivate
$ rmvirtualenv sandbox
```

现在，你唯一应该拥有的环境为 testprojects。

D.9 高级设置回顾

你的计算机现在已经设置好，可以运行高级 Python 库。在与命令行交互和安装包时，你应该感到很舒服。如果你还不是很熟练，我们同样建议你查看附录 C 来学习更多使用命令行的知识。

表 D-1 列出了你在虚拟环境中最常用的命令。

表D-1：回顾命令

命令	动作
`mkvirtualenv`	创建一个环境
`rmvirtualenv`	移除一个环境
`workon`	激活一个环境
`deactivate`	禁用当前活跃的环境
`pip install`	在活跃环境中安装包[a]
`pip uninstall`	在一个活跃的环境中卸载包[b]
`pip freeze`	返回一个活跃环境中已安装库的列表

a 如果没有环境是活跃的，库会被安装到你系统上 Python 的第二个副本上，它通过 Homebrew 安装。你的系统 Python 不会受到影响。

b 见前一条脚注。

附录E

Python陷阱

同你学习的其他语言一样，Python 也有它的怪癖和特点。其中一些特点是脚本语言共有的，如果你有脚本语言的经验，可能不会感到惊讶。其他的特点是 Python 独有的。我们总结了其中一些特点，但并不完全，所以你可以自己去了解更多。我们希望本附录可以帮助你调试代码，也能让你了解 Python 行为方式的原因。

E.1　空白

可能你已经注意到，Python 使用空白作为代码结构的一部分。空白被用来缩进函数、方法和类；去执行 if-else 语句；创建持续的代码行。在 Python 中，空白是一种特殊的操作符，能帮助转化 Python 代码为可执行的代码。

下面是 Python 文件中空白的一些最佳实践。

- 不要使用 tab，使用空格。
- 对于每一个缩进块，使用 4 个空格。
- 为悬挂式缩进选择一种好的缩进策略（可以通过一个分隔符、一个额外的缩进或一个单个缩进对齐，但是必需选用可读性和实用性最好的方式；参见 PEP-8，https://www.python.org/dev/peps/pep-0008/#indentation）。

> PEP-8（或 Python 增强方案 #8）是一个 Python 风格指南，给出了缩进的最佳实践，并针对如何命名变量、代码行换行、格式化代码提出了建议，让代码可读、易于使用、易于分享。

如果你的代码没有正确地缩进，并且 Python 不能解析你的文件，你会得到一个 `IndentationError`。错误信息会告诉你哪一行代码没有正确缩进。在你最喜欢的文本编辑器中设置 Python 的语法提示器是相当简单的，以便在你工作时自动检查你的代码。举个例子，对于 Atom，有一个很棒的 PEP-8 提示器（https://atom.io/packages/linter-python-pep8）。

E.2 可怕的GIL

GIL（Global Interpreter Lock，全局解释器锁）是 Python 解释器用于一次只用一个线程执行代码的一种机制。这意味着当你运行 Python 脚本时，即使上在一台多进程机器上，你的代码也会线性执行。这个设计最初的目的是让 Python 可以通过 C 代码快速地运行，但是仍然是线程安全的。

GIL 给 Python 带来的限制意味着在标准解释器中，Python 从来不会真正地并行化。这对于一些高 I/O 的应用程序，或者严重依赖多重处理的应用程序来说，是一个劣势。[1] 有些 Python 库通过使用多重处理或异步服务 [2]，规避了这些问题，但是它们没有改变 GIL 仍然存在的事实。

即便如此，有很多 Python 核心开发者意识到由 GIL 带来的问题，还有它的好处。在 GIL 成为开发痛点的情况下，通常有不错的应对方案，而且根据你的需要，还有用 C 以外的其他语言编写的其他解释器可用。如果你发现 GIL 成为了代码中的一个问题，很可能你可以重新架构你的代码，或利用一个不同的代码基（例如 Node.js）来满足你的需求。

E.3 =、==与is，以及何时只是复制

在 Python 中，看似相似的函数间有一些重大区别。我们已经了解了一些，但是让我们重新看一下一些代码和输出（使用 IPython）：

```
In [1]: a = 1 ❶

In [2]: 1 == 1 ❷
Out[2]: True

In [3]: 1 is 1 ❸
Out[3]: True

In [4]: a is 1 ❹
Out[4]: True

In [5]: b = []

In [6]: [] == []
Out[6]: True

In [7]: [] is []
Out[7]: False

In [8]: b is []
Out[8]: False
```

注 1：关于 GIL 可视化的更多信息，查看 David Beazley 的“一个可缩放、可交互的 Python 线程可视化”（http://www.dabeaz.com/GIL/gilvis/）。

注 2：关于这些包的功能，查看 Jeff Knupp 关于如何减轻 GIL 问题影响的文章（https://www.jeffknupp.com/blog/2013/06/30/pythons-hardest-problem-revisited/）。

❶ 设置变量 a 为 1。
❷ 检查 1 是否等于 1。
❸ 检查 1 是否与 1 是相同的对象。
❹ 检查 a 是否与 1 是相同的对象。

如果在 IPython 中执行这些代码（这样你可以看到类似于这里展示的输出），你会注意到一些有趣的、可能意外的结果。对于一个整数，我们看到很容易通过多种方式来确定相等。然而对于列表对象，我们发现 is 与其他比较操作符表现得不同。在 Python 中，内存管理操作不同于其他语言。在 Sreejith Kesavan 的博客上（http://foobarnbaz.com/2012/07/08/understanding-python-variables/），有一篇使用可视化技术的文章，讨论了 Python 如何管理内存中的对象。

为了从另外一个视角观察，让我们看一下对象内存的位置：

```
In [9]: a = 1

In [10]: id(a)
Out[10]: 14119256

In [11]: b = a ❶

In [12]: id(b) ❷
Out[12]: 14119256

In [13]: a = 2

In [14]: id(a) ❸
Out[14]: 14119232

In [15]: c = []

In [16]: id(c)
Out[16]: 140491313323544

In [17]: b = c

In [18]: id(b) ❹
Out[18]: 140491313323544

In [19]: c.append(45)

In [20]: id(c) ❺
Out[20]: 140491313323544
```

❶ 将 a 赋值给 b。
❷ 当在这里使用 id 方法时，我们发现 b 和 a 都使用了内存中相同的位置，也就是说，它们在内存中是相同的对象。
❸ 当在此时调用 id 方法时，我们发现 a 在内存中拥有了新的地址。这个地址现在保存着值 2。
❹ 在列表中，可以看到我们在赋值后，列表拥有与赋值对象相同的 id。
❺ 当改变这个列表时，我们发现我们不能改变内存中的位置。Python 列表表现得与整数和字符串略有不同。

这里，我们的目的不是让你对 Python 中的内存分配有深入的理解，而是意识到我们可能不会总是去思考我们到底赋值了什么。在处理列表和字典时，我们需要知道和理解的是，在我们将它赋值为一个新变量的时候，新的变量和旧的变量仍然是内存中的相同对象。如果我们改变其中一个，也改变了另一个。如果只想要改变其中一个或另一个，或者需要创建一个新对象作为对象的副本，需要使用 copy 方法。

让我们通过最后一个示例来解释 copy 与赋值：

```
In [21]: a = {}

In [22]: id(a)
Out[22]: 140491293143120

In [23]: b = a

In [24]: id(b)
Out[24]: 140491293143120

In [25]: a['test'] = 1

In [26]: b ❶
Out[26]: {'test': 1}

In [27]: c = b.copy() ❷

In [28]: id(c) ❸
Out[28]: 140491293140144

In [29]: c['test_2'] = 2

In [30]: c ❹
Out[30]: {'test': 1, 'test_2': 2}

In [31]: b ❺
Out[31]: {'test': 1}
```

❶ 在这行代码中，我们看到，当我们修改 a 时，同样修改了 b，因为它们存储在内存中相同的位置。
❷ 使用 copy 方法我们创建了一个新的变量，c，这是第一个字典的副本。
❸ 这行代码中，我们看到 copy 创建了一个新的对象。它有一个新的 id。
❹ 在修改了 c 之后，我们看到它现在保存着两个键和值。
❺ 即使在 c 修改之后，我们看到 b 仍然是相同的。

在最后这个示例中，很显然，如果你真的想要一个字典或列表的副本，需要使用 copy 方法。如果你想要相同的对象，那么可以使用 =。类似地，如果你想要知道两个对象是不是“相等的”，可以使用 ==，但是如果你想知道它们是否是相同的对象，则使用 is。

E.4 默认函数参数

有时你会想要传递默认变量到你的 Python 函数和方法中。为此，你需要充分理解 Python

何时以何种方式调用这些方法。让我们看一下：

```
def add_one(default_list=[]):
    default_list.append(1)
    return default_list
```

现在，让我们通过 IPython 研究：

```
In  [2]: add_one()
Out [2]: [1]

In  [3]: add_one()
Out [3]: [1, 1]
```

你可能希望每一个函数调用会返回一个新的列表，只包含一个元素，1。相反，两个调用修改相同的列表对象。而实际上，默认参数在脚本第一次解释的时候被声明。如果每次你都想要一个新的列表，可以像下面一样重写函数。

```
def add_one(default_list=None):
    if default_list is None:
        default_list = []
    default_list.append(1)
    return default_list
```

现在我们的代码表现得同我们期望的一样：

```
In  [6]: add_one()
Out [6]: [1]

In  [7]: add_one()
Out [7]: [1]

In  [8]: add_one(default_list=[3])
Out [8]: [3, 1]
```

现在你对内存管理和默认变量有了一些了解，你可以使用你的知识来确定何时检查，以及何时在函数与可执行代码中赋值。深入理解了 Python 何时以何种方式定义对象，我们可以确保这些“陷阱”类型不会给我们的代码添加 bug。

E.5　Python作用域与内置函数：变量名称的重要性

在 Python 中，作用域的执行与你的预期有些许不同。如果你在函数作用域中定义一个变量，这个变量不被函数之外所知。让我们看一下：

```
In [10]: def foo():
   ....:     x = "test"

In [11]: x
.---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-94-009520053b00> in <module>()
----> 1 x
NameError: name 'x' is not defined
```

然而，如果之前定义了 x，我们会得到旧的定义：

```
In [12]: x = 1

In [13]: foo()

In  [14]: x
Out [14]: 1
```

这些与内置函数和方法相关。如果你不小心重写了它们，从那一刻之后你都不能再使用它们了。所以，如果你重写特殊的词列表（list）或日期（date），拥有这些名字的内置函数不会在剩余的代码中正常执行（或者从那一刻之后）：

```
In [17]: from datetime import date

In [19]: date(2015, 2, 5)
Out[19]: datetime.date(2015, 2, 5)

In [20]: date = 'my date obj'

In [21]: date(2015, 2, 5)
.--------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-105-7f129d4341d0> in <module>()
----> 1 date(2015, 2, 5)

TypeError: 'str' object is not callable
```

正如你所见，使用共享名称的变量（或与任何其他标准 Python 命名空间或你使用的任何其他库共享名称）可能造成调试的噩梦。如果你在代码中使用特殊的名称，并且意识到固有的变量或模块名称，就不会花几个小时调试命名空间问题。

E.6　定义对象与修改对象

在 Python 中，定义一个新的对象与修改一个老对象，执行的方式略有区别。假设你有一个函数，为一个整数加 1：

```
def add_one_int():
    x += 1
    return x
```

如果你尝试运行这段函数，应该会收到一个错误：`UnboundLocalError: local variable 'x' referenced before assignment`。然而，如果你在函数中定义了 x，会看到不同的结果：

```
def add_one_int():
    x = 0
    x += 1
    return x
```

这段代码有些复杂（为什么我们不能直接返回 1？），但是这里的重点是，我们在修改变量之前需要先声明变量，即使我们使用了一个看起来像赋值的修改（+=）。在处理像列表

和字典这样的对象工作时留心这一点是特别重要的（因为我们知道修改一个对象会对存储在相同内存位置的其他对象产生副作用）。

需要记住的是，在你想要修改一个对象和想要创建或返回一个新对象时，永远要保持清晰和明确。你命名变量的方式，以及编写与实现函数的方式，是编写清晰与行为可预测的脚本的关键。

E.7 修改不可变对象

想要修改或改变不可变对象时，你需要创建新的对象。Python 不会允许你修改不可变对象，例如元组。在我们讨论 Python 内存管理时，你已经知道一些对象保存在相同的空间中。不可变对象不能被改变，它们总是被重新赋值。让我们看一下：

```
In [1]: my_tuple = (1,)

In [2]: new_tuple = my_tuple

In [3]: my_tuple
Out[3]: (1,)

In [4]: new_tuple
Out[4]: (1,)

In [5]: my_tuple += (4, 5)

In [6]: new_tuple
Out[6]: (1,)

In [7]: my_tuple
Out[7]: (1, 4, 5)
```

可以看到，我们尝试使用 += 操作符去修改原始的元组，并且我们能够成功地做到这一点。然而，我们得到的是一个包含原始元组和追加了元组（4, 5）的新对象。我们最终没有改变 new_tuple 变量，因为我们只是将内存中的一个新地址赋给新的对象。如果你在查看 += 操作之前和之后查看内存地址，你会看到它的改变。

关于不可变对象需要记住的重点是，当改变它们的时候，它们不会使用内存中相同的地址；如果你修改它们，你实际上在创建全新的对象。如果你使用一个拥有不可变对象的类的方法或属性，尤其要记住这一点，因为你要确保自己知道何时在修改它们，何时在创建新的不可变对象。

E.8 类型检查

Python 允许简单的类型转换，这意味着你可以将字符串转换为整数或将列表转换为元组，等等。但是这些动态类型意味着可能会引发问题，特别是在大型的代码仓库中，或在你使用新的库时。常见问题是一些特定的函数、类或方法对应着一些特定的对象类型，而你传递了错误的类型。

随着你的代码变得更高级和复杂，问题会变得越来越难办。由于你的代码更加抽象，你会将所有的代码保存在变量中。如果一个函数或方法返回一个不符预期的类型（例如 None 而不是列表），这个对象可能被传递到另一个函数中——可能是一个不接受 None 类型的函数，之后抛出一个错误。很可能错误被捕获了，但代码会认为异常是因为另外的问题触发的，并且继续执行。这会很快地脱离你的控制，并且变成一个相当难以调试的问题。

对于如何处理这些问题，最好的建议是编写非常精准又清晰的代码。你应该积极测试你的代码（确保没有 bug），持续关注你的脚本，并注意任何反常的行为，以此确保函数永远返回期待的内容。你还需要添加日志来帮助确认对象包含的内容。除此之外，清楚你捕获的异常，而不只是捕获所有的异常，这会帮助你更容易地找到和修复问题。

最后，有时 Python 会实现 PEP-484（https://www.python.org/dev/peps/pep-0484/），它包含了类型提示，允许你检查传递的变量和代码，以自我检查这些问题。这在未来 Python 3 发布之前可能不会被合并，但是好消息是，这已经在进行当中，你可以期待在未来看到更多的有关类型检查的结构。

E.9 捕获多个异常

随着代码的发展，你会想要在同一行代码中捕获多个异常。举个例子，你可能想要捕获一个 TypeError，同时还有 AttributeError。如果你以为传递的是一个字典，而实际上传递的是一个列表，可能就是这种情况。它可能有一些相同的属性，但是不是所有属性。如果你需要在一行中捕获多个类型的错误，就必需在元组中编写异常，让我们看一下：

```
my_dict = {'foo': {}, 'bar': None, 'baz': []}

for k, v in my_dict.items():
    try:
       v.items()
    except (TypeError, AttributeError) as e:
        print "We had an issue!"
        print e
```

你应该会看到下面的输出（很可能呈现顺序不同）：

```
We had an issue!
'list' object has no attribute 'items'
We had an issue!
'NoneType' object has no attribute 'items'
```

我们的异常成功地捕获了两个错误并执行了异常代码块。正如你所见，意识到你需要捕获的错误的类型，并且理解语法（将其放到元组中）对你的代码来说是必需的。如果你简单地列出它们（通过一个列表或只是通过逗号分隔），你的代码可能不会正常地执行，而且你不会捕获到这两个异常。

E.10 调试的力量

随着你成为一名更加高级的开发者和数据处理者，你会遇到更多的问题和错误来调试。我

们希望可以告诉你它会变得更简单，但是在它变简单之前，你的调试过程会更加集中和严谨。这是因为你会用到更高级的代码和库，处理更加困难的问题。

即便如此，你拥有很多技术和工具，可以帮助你脱离困境。你可以在 IPython 中执行代码，在开发过程中得到更多反馈。你可以添加日志到脚本中，以更好地理解发生了什么。如果解析网页时遇到问题，你可以让抓取器截屏，并将它们保存到文件中。你可以在 IPython notebook 中与其他人分享代码，或者在其他许多有帮助的站点分享代码，以得到反馈。

Python 中同样有一些很棒的调试工具，包括 pdb（https://docs.python.org/2/library/pdb.html），它允许你逐句执行代码（或模块中的其他代码），并且在所有的错误前后，精确地看到每个对象保存的内容。YouTube 上有一个很棒的关于 pdb 的快速介绍（https://www.youtube.com/watch?v=bZZTeKPRSLQ），展示了一些在代码中使用 pdb 的方式。

除此之外，你需要阅读并编写文档和测试。在本书中我们已经介绍了一些基础，但是我们强烈建议你将本书作为一个起点，并在将来进一步研究文档和测试。Ned Batchelder 最近关于上手测试的 PyCon 讲座（https://www.youtube.com/watch?v=FxSsnHeWQBY）是一个好的起点。Jacob Kaplan-Moss 在 PyCon 2011 上也做了一个很棒的关于文档的讲座（https://www.youtube.com/watch?v=z3fRu9pkuXE）。通过阅读和编写文档，以及编写和执行测试，你可以确保没有因为错误的信息而将错误引入代码，或没有因为未做测试而遗留错误。

我们希望本书是这些概念的优秀入门介绍，但是我们鼓励你继续阅读和开发，通过寻找更多的 Python 学习资源，成为一名更出色的 Python 开发者。

附录F

IPython 指南

尽管你的 Python shell 是有用的，但它缺少了 IPython（http://ipython.org/）的很多魔力。IPython 是一个增强的 Python shell，提供了一些易于使用的快捷方式和在 shell 环境中同 Python 交互的额外能力。它最初由想要一个更简单的 Python shell 的科学家和学生开发（http://blog.fperez.org/2012/01/ipython-notebook-historical.html）。后来，它变成了学习 Python 以及通过解释器与 Python 交互的事实上的标准。

F.1　为什么使用IPython

IPython 提供了很多在标准 Python shell 中缺少的功能。安装和使用 IPython 作为你的 shell 有很多好处。它的特性包括：

- 易于阅读的文档钩子
- 自动补全和用于库、类和对象探索的魔法命令
- 内联图片和图表生成
- 用来浏览历史、创建文件、调试脚本、重新加载脚本等的辅助工具
- 内置的命令行工具帮助
- 启动时的自动导入

它也是 Jupyter（https://jupyter.org/）的一个重要组成部分。Jupyter 是一个共享的 notebook 服务器，可以快速方便地在浏览器中进行数据探索。我们在第 10 章中介绍了使用 Jupyter 进行代码分享和演示。

F.2　IPython起步

IPython 可以用 pip 方便地安装：

```
pip install ipython
```

如果你正在使用多个虚拟环境，可能想要安装全局 IPython，或在每个虚拟环境中都安装一个。为了开始使用 IPython，直接在终端窗口中输入 `ipython`。你应该会看到一个类似这样的提示：

```
$ ipython
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
In [1]:
```

现在你可以输入 Python 命令，就像在普通的 Python shell 中一样。例如：

```
In [1]: 1 + 1
Out[1]: 2

In [2]: from datetime import datetime

In [3]: datetime.now()
Out[3]: datetime.datetime(2015, 9, 13, 11, 47, 49, 191842)
```

当需要退出 shell 时，你可以输入 `quit()`、`exit()`，或者在 Windows/Linux 上输入 Ctrl-D，在 Mac 上输入 Cmd-D。

F.3 魔法函数

IPython 有大量所谓的*魔法*函数，能在探索和编程的时候帮助你。这里有一些非常重要的函数，特别是对于初级开发者来说。

为了看到你已经导入和活动的对象，可以输入 `%whos` 或 `%who`。让我们看一下它们的用法：

```
In [1]: foo = 1 + 4

In [2]: bar = [1, 2, 4, 6]

In [3]: from datetime import datetime

In [4]: baz = datetime.now()
In [5]: %who
bar baz datetime    foo

In [6]: %whos
Variable   Type        Data/Info
--------------------------------
bar        list        n=4
baz        datetime    2015-09-13 11:53:29.282405
datetime   type        <type 'datetime.datetime'>
foo        int         5
```

在你忘记了变量名称或者想要通过一个简洁的列表看到在变量中存储的内容时，这非常有用。

另外一个有用的工具是快速查找与库、类或对象相关的文档的能力。如果你在方法、类、库或属性名称的最后输入一个 ?，IPython 会尝试获取所有相关的文档，内联地展示它。举个例子：

```
In [7]: datetime.today?
Type:       builtin_function_or_method
String Form:<built-in method today of type object at 0x7f95674e0a00>
Docstring:  Current date or datetime:
            same as self.__class__.fromtimestamp(time.time()).
```

有大量与此类似的 IPython 扩展和函数，对开发极其有用，特别是随着你作为一名开发者不断成长，碰到更加复杂的问题时。表 F-1 列出了最为有用的几个工具，网络上还有一些非常棒的演讲稿和会议演讲（http://ipython.org/presentation.html）以及交互示例（http://nbviewer.jupyter.org/github/ipython/ipython/blob/master/examples/Index.ipynb），此外还有关于库的非常棒的文档（http://ipython.org/documentation.html）。

所有的 IPython 扩展必需在 IPython 会话的开始使用 `%load_ext extension_name` 加载。如果你想要安装额外的扩展，GitHub 上有一个非常棒的可用扩展及其使用方式的列表（https://github.com/ipython/ipython/wiki/Extensions-Index）。

表F-1：有用的IPython扩展与函数

命令	描述	目的	文档
`%autoreload`	允许你只通过一次调用重新加载所有导入的脚本	在编辑器中修改脚本、在 IPython shell 中调试脚本，进行活跃开发的时候，非常有帮助	http://ipython.org/ipython-doc/dev/config/extensions/autoreload.html
`%store`	允许你存储保存的变量，在下一个会话中使用	在需要保存一些经常使用的变量，或者你的工作被中断因而需要保存当前的工作供以后使用时，非常有帮助	http://ipython.org/ipython-doc/dev/config/extensions/storemagic.html
`%history`	打印你的会话历史	展示你已经运行过的命令的输出	https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-history
`%pdb`	用于过程较长的调用的交互式调试模块	强有力的调试库，在导入了较长的脚本或模块时特别有用	https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-pdb
`%pylab`	引入 `numpy` 和 `matplotlib` 来与你的会话交互式工作	允许你在 IPython shell 中使用统计和图表工具	https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-pylab
`%save`	保存你的会话历史到一个输出文件	如果你花费了很长的时间调试，这是一个开始编写脚本的好方式	https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-save
`%timeit`	计算代码中一行代码或多行代码的执行时间	用于对 Python 脚本和函数进行性能调优	https://ipython.org/ipython-doc/dev/interactive/magics.html#magic-timeit

还有很多可用的魔法命令（https://ipython.org/ipython-doc/dev/interactive/magics.html）。它们的有用性取决于你在开发中使用 IPython 的方式，但是随着你作为一名开发者不断成长，使用它们会通过 IPython 为你简化其他的任务。

F.4 最后的思考：一个简单的终端

无论是只在一个 notebook 中，还是在一个活跃的终端开发中使用 IPython，我们相信它会帮助你编写和理解 Python，并且帮助你成长为一个优秀的开发者。你早期的开发主要是探索 Python 如何工作，以及在开发过程中会遇到的错误和异常。IPython 对这些任务很在行，因为你可以在下一行输入中重试代码。我们希望 IPython 会在未来数年伴随你学习和编写 Python。

附录 G

使用亚马逊网络服务

如果开始为数据处理需求使用亚马逊和亚马逊云服务，你首先需要一个设置好以供使用的服务器。我们会学习如何让你的第一台服务器就绪并且运行。

在第 10 章，我们介绍了 AWS 之外的一些选择，包括 DigitalOcean、Heroku、GitHub Pages，以及使用一个托管服务提供商。根据你对不同部署与服务器环境的兴趣，建议使用多个选项，然后选择最适合你的选项。

AWS 作为第一个云平台而流行，但是它同样会带来许多困扰。我们想要包含一个教程来帮助你浏览整个过程。我们同样强烈建议使用 DigitalOcean 作为进入云的开始；它们的教程（https://www.digitalocean.com/help/getting-started/setting-up-your-server/）和指引（https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-14-04）是非常有帮助的。

G.1　启动AWS服务器

为了启动一个服务器，在 AWS 控制台（https://console.aws.amazon.com）选择“Compute”下的“EC2”（你需要登录或创建一个账户来访问控制台）。这会带你来到 EC2 着陆页（https://console.aws.amazon.com/ec2/v2/home）。在这里，点击“Launch Instance”按钮。

这时，你会开始跟随一个教程来设置你的实例。你在这里选择的所有东西都是可编辑的，所以不知道选择什么也不必担心。这本书提供了以廉价又快速的方式设置并运行服务器的建议，但这不意味着这就是你需要的解决方案。如果你碰到了空间等问题，可能需要一个更大因而也更贵的配置 / 实例。

在下面的这一小节里，我们会带着你浏览一遍我们推荐的设置。

G.1.1　AWS步骤1：选择一个亚马逊机器镜像（AMI）

机器镜像基本上是一个操作系统镜像（或快照）。最普遍的操作系统是 Windows 和 OS X。然而，基于 Linux 的系统通常用作服务器。我们推荐最新的 Ubuntu 系统，在本书编写时的版本为“Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - amid05e75b8”。

G.1.2　AWS步骤2：选择一个实例类型

实例类型是你启动的服务器的容量。选择“t2.micro (Free tier eligible)”。不要扩容，除非你明确需要，因为这样会浪费钱。为了学习更多关于实例的知识，查看 AWS 关于实例类型（https://aws.amazon.com/ec2/instance-types/）与价格（https://aws.amazon.com/ec2/pricing/）的文章。

选择“Review and Launch”，这将带你到第 7 步。

G.1.3　AWS步骤7：学习实例启动

在页面的顶部，你会注意到一条信息：“提高你的实例安全。你的安全组，launch-wizard-4，正对全世界开放。”对于真正的产品实例，或带有敏感信息的实例，强烈建议提高安全，同时采用其他的安全措施。查看 AWS 的文章“加强你的 EC2 实例安全的建议”（https://aws.amazon.com/articles/1233/）。

G.1.4　AWS额外问题：选择一个存在的键对或创建一个新的

一个键对类似于一个服务器的键集合，这样服务器知道谁有权利使用。选择“Create a new key pair”并且给它命名。我们已经命名我们的实例为 data-wrangling-test，但是你可以给它取任何你可以识别的名字。当你完成时，下载键对到一个你可以在随后找到的地方。

最后，点击“Launch Instances”。启动实例后，你会在屏幕上得到一个实例 ID。

> 如果你担心服务器花销，在 AWS 首选项中创建一个账单报警（https://console.aws.amazon.com/billing/home?#/preferences）。

G.2　登录AWS服务器

为了登录到服务器上，你需要进入 AWS 控制台的实例来得到更多的信息。在控制台中，选择 EC2，之后选择“1 Running Instance”（如果你有不止一个实例，数字会增大）。你会看到你的服务器列表。除非你提供过名称，否则你的服务器不会拥有名称。通过点击列表上的空框，为你的实例命名。我们将其命名为 data-wrangling-test，以便持久化。

为了登录到我们的服务器上，我们会跟随一篇关于连接到 Linux 实例的 AWS 文章（http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html#ec2-connect-to-instance-linux）的指示。

G.2.1　得到实例的公共DNS名称

公共 DNS 名称是实例的 Web 地址。如果你有一个看起来像 Web 地址的值，继续阅读下一个小节。如果值为“--”，那么你需要跟随下面这些额外的步骤（来自 Stack Overflow ，http://stackoverflow.com/questions/20941704/ec2-instance-has-no-public-dns/26403671#26403671）。

(1) 访问 console.aws.amazon.com。

(2) 访问 Services（顶部导航）→ VPC（接近列表的末尾）。

(3) 打开你的 VPC（左侧栏目）。

(4) 选择连接到你的 EC2 的 VPC。

(5) 在“Actions”下拉菜单，选择“Edit DNS Hostnames”。

(6) 修改“Edit DNS Hostnames”为“Yes”。

如果你返回 EC2 实例，应该看到它现在有了一个公共 DNS 名称。

G.2.2　准备你的私钥

你的私钥是一个下载的 .pem 文件。将它移动到一个你了解并且能够记住的文件夹是好的想法。对于基于 Unix 的系统，你的密钥会保存在 home 目录下名为 .ssh 的文件夹。对于 Windows，默认的位置是 C:\Documents and Settings\.ssh\ 或 C:\Users\.ssh。你需要复制你的 .pem 文件到这些文件夹。

然后，你需要运行 `chmod` 命令来改变 .pem 文件的权限为 `400`。修改权限为 `400` 意味着，文件只能被所有者访问。这保证了在多账户计算机环境下文件的安全：

```
chmod 400 .ssh/data-wrangling-test.pem
```

G.2.3　登录你的服务器

现在，你已经有了所有登录服务器所需的一切。运行下面的命令，但是用你的键对替换 *`my-key-pair.pem`*，同时用你的公开 Web 地址替换 *`public_dns_name`*：

```
ssh -i ~/.ssh/my-key-pair.pem_ ubuntu@_public_dns_name
```

例如：

```
ssh -i data-wrangling-test.pem ubuntu@ec2-12-34-56-128.compute-1.amazonaws.com
```

当提示“Are you sure you want to continue connecting (yes/no)?”时，输入 `yes`。

现在，你的提示会有一些轻微的改变，说明你正位于你创建的控制台中。你现在可以继续设置你的服务器，将代码部署到服务器上，设置在服务器上运行的自动化程序。在第 14 章你可以阅读更多关于部署代码到新服务器上的知识。

为了退出服务器，输入 Ctrl-C 或 Cmd-C。

G.3 小结

现在你已经有了第一个设置好并且运行起来的 AWS。使用在第 14 章学到的知识将代码部署到服务器，并且立即运行你的数据处理程序。

关于作者

Jacqueline Kazil 是一名数据爱好者。在职业生涯中，她曾在专注于金融、政府和新闻业的技术公司工作。最值得一提的是，她是一位前总统创新研究员（Presidential Innovation Fellow，https://presidentialinnovationfellows.gov/fellows/），还联合创建了一家名为 18F 的政府技术组织（https://18f.gsa.gov/）。她在职业生涯中完成了许多数据科学与数据处理项目，包括 Geoq（https://geo-q.com/geoq/，一个开源的绘图工作流程工具）、Congress.gov（https://www.congress.gov/）网站改造与美国最高机密项目（Top Secret America，http://projects.washingtonpost.com/top-secret-america/）。她活跃于 Python 与数据的社区——Python 软件基金会（https://www.python.org/psf/）、PyLadies（http://www.pyladies.com/）、Women Data Science DC（http://www.meetup.com/WomenDataScientistsDC/）等。她在华盛顿哥伦比亚特区的聚会、会议和迷你训练营上教人学习 Python。她还经常与好友 Ellie（https://twitter.com/ellie_the_brave）一起结对编程。你可以在 Twitter 上找到她（https://twitter.com/jackiekazil），也可以关注她的博客 The coderSnorts（https://medium.com/coder-snorts）。

Katharine Jarmul 是一名 Python 开发者，她喜欢数据分析和获取、网页抓取、教人学习 Python 以及与 Unix 有关的一切。她曾在大大小小的创业公司工作，然后开始了海外咨询业务。最初在洛杉矶，她于 2008 年在《华盛顿邮报》工作时学习了 Python。作为 PyLadies 的创始人之一，Katharine 希望通过教育和培训来促进 Python 和其他开源语言的多元化。她领导了许多研讨会与辅导班，其内容从 Python 入门主题到 Python 高级话题。想进一步了解今后的培训项目，请通过她的 Twitter 账号（https://twitter.com/kjam）或网站（http://kjamistan.com/）与她联系。

关于封面

本书封面上的动物是一只蓝唇树蜥（blue-lipped tree lizard，拉丁名为 Plica umbra）。Plica 属的成员都是中等大小，尽管它们属于俗称的新热带地蜥（neotropical ground lizard）科，但主要生活在南美洲和加勒比地区的树上。蓝唇树蜥主要以蚂蚁为食，并且是 Plica 属中唯一不以脖子上的一束刺为特征的物种。

O'Reilly 图书封面上的很多动物都是濒危动物，它们对世界都很重要。想要详细了解如何帮助这些动物，请访问 animals.oreilly.com。

封面图片来自于 Lydekker 的 *Natural History*。

O'REILLY®

Python数据处理

用传统的电子表格来处理数据不仅效率低下，而且无法处理某些格式的数据，对于混乱或庞大的数据集更是束手无策。本书将教你如何利用语法简单、容易上手的Python轻松处理数据。作者通过循序渐进的练习，详细介绍如何有效地获取、清洗、分析与呈现数据，如何将数据处理过程自动化，如何安排文件编辑与清洗任务，如何处理更大的数据集，以及如何利用获取的数据来创作引人入胜的故事。学完本书，你的数据处理和分析能力将更上一层楼。

- **快速了解Python基本语法、数据类型和语言概念**
- **概述数据的获取与存储方式**
- **清洗数据并格式化，以消除数据集中的重复值与错误**
- **学习何时对数据进行标准化，何时对数据清理进行测试并将其脚本化**
- **使用Scrapy写网络爬虫**
- **利用新的Python库和技术对数据集进行探索与分析**
- **使用Python解决方案将整个数据处理过程自动化**

Jacqueline Kazil，数据科学家，资深软件开发者。活跃于Python软件基金会、PyLadies等社区。曾参与美国总统创新伙伴项目，是美国政府技术组织18F的联合创始人。曾担任《华盛顿邮报》数据记者。

Katharine Jarmul，资深Python开发者，PyLadies联合创始人。喜欢数据分析和获取、网页抓取、教人学习Python以及Unix，期望通过教育和培训来促进Python和其他开源语言的多元化。

“如果你一直感觉电子表格（甚至关系型数据库）无法回答你想要提出的问题，或者除这些工具之外你准备进一步学习，那么这本书非常适合你。我一直在等待这本书的出现。”

——Derek Willis

ProPublica新闻应用开发者

OpenElections联合创始人

“所有新手数据科学家、数据工程师或其他技术方面的数据专家都应该读一读这本实践指南。数据处理领域正需要这样一本书，真希望我第一次开始用Python处理数据时就能有它指导。”

——Tyrone Grandison博士

Proficiency Labs Intl. CEO

DATABASES

封面设计：Randy Comer　张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　计算机／数据分析

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

ISBN 978-7-115-45919-0

ISBN 978-7-115-45919-0
定价：99.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks